الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي

Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة

Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا

Faculté de Technologie

قسم الإلكترونيك

Département d'Électronique

Mémoire de Master

En Electronique

Spécialité : Électronique des systèmes embarqués

Présenté par :

KAHLIS MAHDI

&

BADA SAMI SALAH EDDINE

Développement d'un système de surveillance et de contrôle agricole basé sur l'ESP32, un serveur TCP et une application mobile Android

Encadrée par : Mme NACEUR DJAMILA

Année Universitaire 2024 – 2025

Remerciements

Avant tout, nous exprimons notre profonde gratitude à Dieu Tout-Puissant, qui nous a donné la force, la santé et la patience nécessaires pour mener à bien ce travail.

Nous tenons à remercier sincèrement nos parents pour leur amour, leurs sacrifices et leur soutien inconditionnel tout au long de ce parcours. Leur confiance et leurs prières nous ont été d'un grand réconfort.

Nos remerciements vont également à nos enseignants et encadrants, pour leur accompagnement, leurs conseils avisés et leur disponibilité. Leur encadrement a été essentiel à la réussite de ce projet.

Nous remercions aussi nos amis et collègues, pour leur soutien moral, leur aide et les moments de partage qui ont rendu cette expérience plus riche et plus agréable.

Enfin, nous adressons nos plus sincères remerciements à toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Bada sami salah eddine, Kahlis mahdi.

Dédicaces

À nos chers parents,

Pour votre amour inconditionnel, vos sacrifices silencieux et votre soutien constant. Grâce à votre foi en nous et à vos prières, nous avons pu surmonter les difficultés et mener ce projet à son terme. Ce travail est le fruit de votre dévouement et de votre confiance. Nous vous en sommes éternellement reconnaissants.

À nos amis et proches,

Votre soutien, vos encouragements et votre patience ont été essentiels. Merci d'avoir été là, dans les bons comme dans les moments plus difficiles. Votre présence a été précieuse tout au long de cette aventure.

À tous ceux qui, de près ou de loin, ont contribué à ce parcours,

Cette réussite est aussi la vôtre. Vous nous avez donné la motivation de persévérer et de croire en nous. Merci de tout cœur.

Bada sami salah eddine, Kahlis mahdi.

ملخص:

يعتمد على مجموعة من الحساسات البيئية)درجة الحرارة، الرطوبة، (serre يعرض هذا المشروع تصميم نظام ذكي لبيت زجاجي 32ESP. الضوء، رطوبة التربة(ومجموعة من المشغلات)مضخة، سخان، مروحة، إضاءة(يتم التحكم بها بواسطة المتحكم الدقيق ، وتطبيق Flask ، خادوم ويب باستخدامMySQL ، قاعدة بياناتPython بلغة TCP يعتمد النظام على بنية طبقية تشمل خادوم أندرويد مخصص للمراقبة والتحكم اللحظي. يتم تبادل البيانات عبر رموز عمليات وبروتوكولات اتصال شبكي موثوقة. يتميز التطبيق

المحمول بواجهة استخدام بديهية تضم رسومًا بيانية تفاعلية وأزرار تحكم ذكية. أظهرت النتائج فعالية النظام في ظروف تشغيل حقيقية، مما .يفتح آفاقًا مستقبلية نحو أتمتة البيوت الزجاجية وإدماج تقنيات الذكاء الاصطناعي

الكلمات المفتاحية: الزراعة الذكية، بيت زجاجي متصل، أجهزة إنترنت الأشياء، تطبيق أندرويد

Résumé : Ce mémoire présente la conception d'un système de serre connectée combinant des capteurs environnementaux (température, humidité, lumière, sol) et des actionneurs (pompe, chauffage, ventilateur, lumière) contrôlés via un microcontrôleur ESP32. Le système utilise une architecture en couches avec un serveur TCP sous Python, une base de données MySQL, un serveur web Flask et une application Android pour le contrôle et la visualisation en temps réel. L'échange de données se fait via des codes d'opération et des communications réseau robustes. L'application mobile intègre une interface intuitive avec des graphiques et commandes interactives. Les résultats obtenus montrent un bon fonctionnement du système en conditions réelles. Ce projet ouvre des perspectives vers l'automatisation intelligente de serres et l'intégration future de l'intelligence artificielle.

Mots clés : Agriculture intelligente, Serre connectée, IOT Appareils, Application Android.

Abstract : This thesis presents the design of a smart greenhouse system combining environmental sensors (temperature, humidity, light, soil) and actuators (pump, heater, fan, light) controlled via an ESP32 microcontroller. The system adopts a layered architecture with a Python-based TCP server, a MySQL database, a Flask web server, and an Android application for real-time control and monitoring. Data exchange is handled through operation codes and reliable network communication. The mobile app features an intuitive interface with interactive graphs and controls. The results demonstrate good system performance under real conditions. This project paves the way for smart greenhouse automation and future integration of artificial intelligence.

Keywords : Smart agriculture, Connected greenhouse, IOT devices, Android app.

Listes des acronymes et abréviations

Acronyme / Abréviation	Signification
IoT	Internet of Things (Internet des Objets)
IA	Intelligence Artificielle
ESP32	Microcontrôleur 32 bits avec Wi-Fi et Bluetooth intégré, utilisé pour l'IoT
Li-ion	Lithium-Ion (type de batterie rechargeable)
Li-Po	Lithium-Polymère (type de batterie rechargeable)
USB	Universal Serial Bus (interface de connexion standard)
AC	Alternating Current (Courant alternatif)
DC	Direct Current (Courant continu)
TCP	Transmission Control Protocol (Protocole de Contrôle de Transmission)
UDP	User Datagram Protocol (Protocole de Datagramme Utilisateur)
IP	Internet Protocol (Protocole Internet)
Wi-Fi	Wireless Fidelity (Connexion sans fil à un réseau local)
AP	Access Point (Point d'accès Wi-Fi)
SSID	Service Set Identifier (Nom du réseau Wi-Fi)
EEPROM	Electrically Erasable Programmable Read-Only Memory (Mémoire non volatile reprogrammable)
I2C	Inter-Integrated Circuit (Bus de communication série)
SDA	Serial Data Line (Ligne de données du bus I2C)

SCL	Serial Clock Line (Ligne d'horloge du bus I2C)
GPIO	General Purpose Input/Output (Broche d'entrée/sortie générale d'un microcontrôleur)
API	Application Programming Interface (Interface de Programmation Applicative)
MySQL	Structured Query Language, système de gestion de base de données relationnelle
DB	Database (Base de données)
UI	User Interface (Interface utilisateur)
GUI	Graphical User Interface (Interface Graphique Utilisateur)
Python	Langage de programmation interprété, orienté objet
Flask	Framework web léger en Python
Gemini	API d'intelligence artificielle développée par Google
DHT11	Capteur de température et d'humidité de l'air
BH1750	Capteur de luminosité numérique
DS18B20	Capteur de température numérique (souvent utilisé pour le sol)
YL-69	Capteur d'humidité du sol
op_code	Operation Code (Code d'opération, identifiant le type de message)
pin	Broche (connecteur physique sur un microcontrôleur)
Main.py (etc.)	Fichier source Python (extension .py)
XML	eXtensible Markup Language
NPM	Node Package Manager
SMTP	Simple Mail Transfer Protocol
JS	JavaScript
HTTP	HyperText Transfer Protocol

ID	Identifier
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
HTML	HyperText Markup Language

Table des matières

Liste des figures

Liste des tableaux

Introduction Générale :

L'évolution des technologies embarquées et de l'Internet des Objets (IoT) a profondément transformé de nombreux secteurs, notamment l'agriculture. Aujourd'hui, la demande croissante en solutions intelligentes pour la gestion des ressources et l'optimisation des rendements agricoles pousse les chercheurs et ingénieurs à proposer des systèmes toujours plus autonomes, connectés et efficaces.

Dans ce contexte, le présent travail s'inscrit dans une démarche de modernisation de l'agriculture sous serre, à travers la conception d'un système intelligent de surveillance et de contrôle des paramètres environnementaux. Ce système repose sur une architecture IoT intégrant des capteurs de température, d'humidité, de lumière, ainsi que des actionneurs (pompes, ventilateurs, chauffage), tous connectés à une carte ESP32. L'ensemble communique avec un serveur central, développé en Python, qui centralise les données et coordonne les actions.

Le système développé se compose également d'une application mobile Android, permettant à l'utilisateur de consulter en temps réel l'état de la serre et de contrôler les équipements à distance. Par ailleurs, un site web vitrine réalisé avec React.js permet de valoriser la solution proposée, en expliquant son fonctionnement et ses avantages, et en assurant une interface de présentation professionnelle.

La diversité des modules mis en œuvre – serveur TCP sous Debian, microcontrôleur ESP32, interfaces Android et web, communication en temps réel, base de données MySQL, intégration d'une API d'intelligence artificielle (Gemini) – impose une approche méthodologique rigoureuse et modulaire.

Ainsi, le présent mémoire vise à répondre à la problématique suivante :
Comment concevoir un système IoT intelligent, modulaire et interactif, pour la surveillance et le contrôle à distance d'une serre agricole, tout en assurant une gestion efficace des données et une

Pour répondre à cette question, ce mémoire est structuré en quatre chapitres

- Chapitre 1 : Concepts généraux, présente les fondements de l'IoT, les applications dans l'agriculture intelligente, les types de capteurs utilisés, les microcontrôleurs, les protocoles de communication et les technologies web associées.
- Chapitre 2 : Développement du serveur et du code ESP32**,** décrit l'architecture du serveur, la gestion des communications, le rôle de la base de données, ainsi que le fonctionnement du code embarqué sur la carte ESP32.
- Chapitre 3 : Développement de l'application Android et du site web, détaille la conception des interfaces utilisateurs mobiles et web, l'architecture des composants React, la gestion des événements et l'intégration des services.
- Chapitre 4 : Interaction des interfaces avec le serveur**,** présente les mécanismes d'échange de données entre les différentes parties du système, la communication client-serveur via TCP, l'implémentation des opcodes, ainsi que l'interconnexion entre les fragments Android, le serveur et les équipements connectés.

Enfin, une conclusion générale viendra clore ce mémoire en récapitulant les apports du projet et en ouvrant sur des perspectives futures d'amélioration, notamment en matière d'intelligence artificielle, d'automatisation et de déploiement à grande échelle.

1. Chapitre 1 : Concepts Généraux

1.1. Introduction :

L’internet des Objets (IoT) a profondément transformé le domaine de l’agriculture en apportant des solutions intelligentes qui améliorent l’efficacité, la durabilité et la productivité. Grâce à la connexion entre capteurs et machines, il est désormais possible de surveiller à distance et en temps réel des informations essentielles telles que l’état du sol, les conditions météorologiques ou encore la santé des cultures.

Ces technologies permettent aux agriculteurs de mieux gérer l’irrigation, de contrôler les équipements, et d’optimiser l’utilisation des ressources tout en réduisant les pertes.

Dans cette optique, il est nécessaire de développer un système intelligent basé sur l’IoT, capable de surveiller en continu les paramètres environnementaux d’une serre (température, humidité, lumière, etc.), afin de garantir des conditions de culture optimales et de prévenir les problèmes liés aux changements brusques de l’environnement.

1.2. Internet des Objets

L’internet des objets (IoT, ou Internet of Things en anglais) désigne un réseau d'objets physiques interconnectés capables de collecter, échanger et analyser des données via internet ou d'autres réseaux de communication. L'objectif de l'IoT est de rendre ces objets "intelligents" en leur permettant de communiquer entre eux ou d'agir de manière autonome sans intervention humaine directe. Par exemple, des capteurs de température ou d'humidité peuvent transmettre des données à une application mobile, ou déclencher des actions automatiques, Figure 1.1.

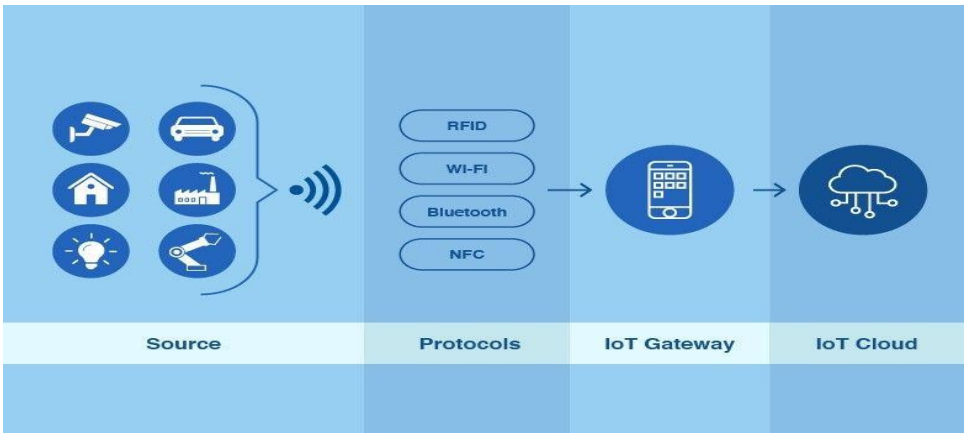


Figure 1.1 : L'architecture de l'IOT.

1.2.1. Agriculture intelligente basée sur l'IoT

Dans cette section, nous présentons un cadre commun pour un écosystème IoT pour l'agriculture intelligente basé sur trois composants principaux, notamment les appareils IoT, les technologies de communication et les solutions de traitement et de stockage de données, Figure1.2.

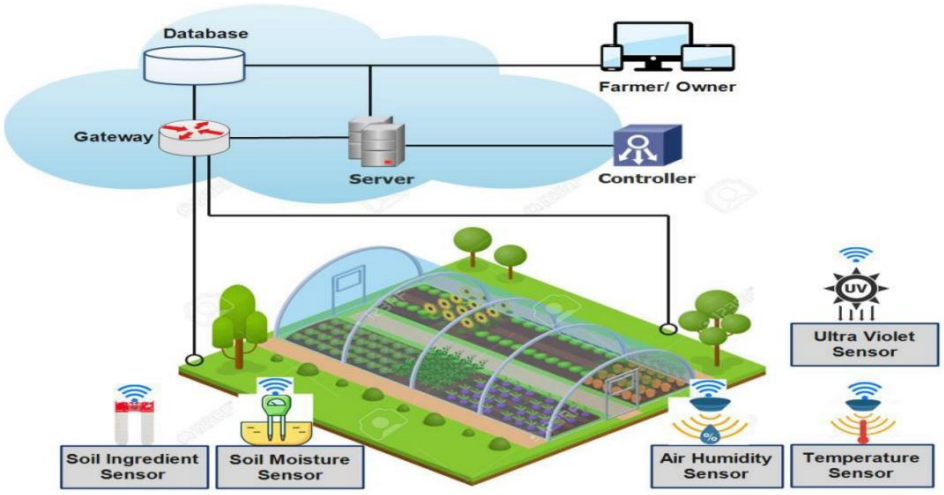


Figure 1.2 : Illustration d'une application IoT pour la surveillance des conditions agricoles dans une serre.

1.2.2. L'IoT moderne dans l'agriculture

L'IOT moderne dans l'agriculture, Figure1.3. désigne l'évolution actuelle des technologies connectées, où des milliards d'objets physiques, sont reliés à L'internet pour collecter, échanger et analyser des données en temps réel. Il se distingue par :

- Une plus grande intelligence **:** grâce à l'intelligence artificielle (IA), les objets peuvent apprendre, prévoir et réagir automatiquement.
- Des connexions plus rapides : Avec la technologie des serveurs, le transfert de données est presque instantané.
- Une meilleure sécurité : les protocoles modernes protègent mieux les données échangées.
- Interopérabilité accrue : les éléments peuvent fonctionner ensemble en même temps.

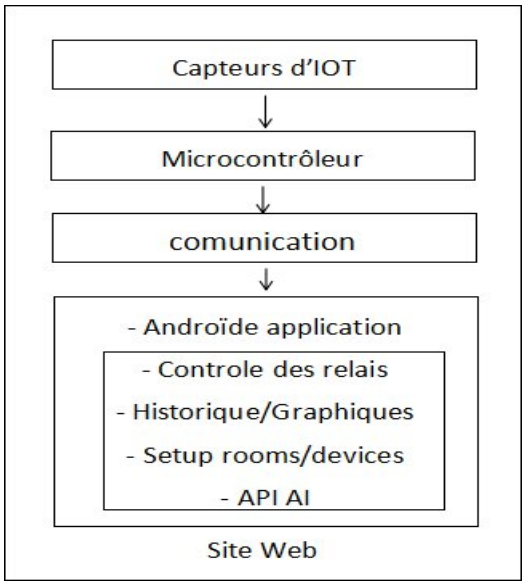


Figure 1.3 : schéma générale de l'IOT moderne.

1.2.3. Appareils IOT dans les serres

L'architecture typique d'un appareil IoT repose sur des capteurs conçus pour collecter des informations sur l'environnement, des actionneurs connectés via des liaisons filaires ou sans fil et un système embarqué intégrant un processeur et des capteurs intelligentes, des modules de communication, des interfaces pour l'application et une alimentation, Figure 1.4.

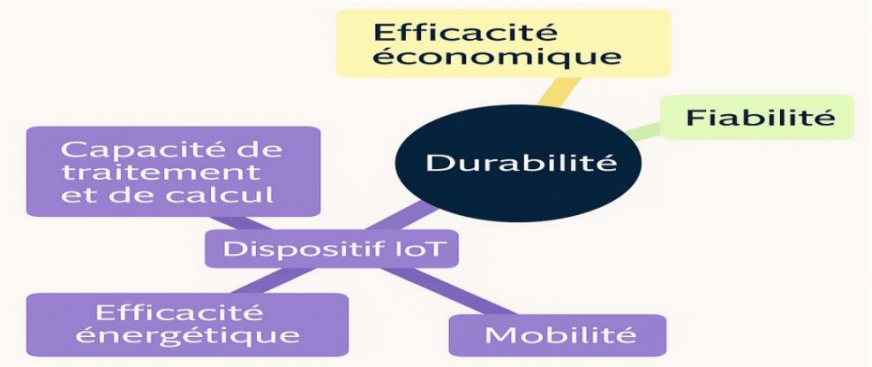


Figure 1.4**:** Principales caractéristiques des appareils IoT

1.2.4. Avantages de l'IOT dans l'agriculture

Les technologies agricoles intelligentes offrent aux agriculteurs une puissante boîte à outils pour améliorer l'efficacité, réduire les déchets et augmenter la rentabilité. À l'aide d'appareils IoT, agriculture peut surveiller la santé des cultures, l'état des sols et les conditions météorologiques en temps réel, ce qui permet une affectation précise des ressources et des interventions en temps opportun.

1.3. Technologies des communications dans le IOT

L'étude des technologies de communication pour l'IoT a montré que pour intégrer l'IoT au secteur de l'agriculture intelligente, les technologies de communication doivent progressivement améliorer l'évolution des appareils IoT. Elles jouent un rôle important dans le développement des systèmes IoT. Les solutions de communication existantes peuvent être classées comme suit : protocole, spectre et topologie.

1.3.1. Protocoles

De nombreux protocoles de communication sans fil ont été proposés pour le secteur de l'agriculture intelligente. Grâce à ces protocoles, les appareils d'un système agricole intelligent peuvent interagir. Les protocoles de communication typiques à faible consommation d'énergie couramment utilisés dans l'agriculture intelligente peuvent être divisés en catégories à courte et longue portée en fonction de la portée de communication, tableau 1.1.

1.3.2. Spectre

Chaque appareil radio utilise certaines bandes de fréquences pour communiquer, les bandes de spectre sans licence pour les opérations sans licence à des fins scientifiques, Ces bandes de spectre sont souvent utilisées pour des niveaux de faible puissance et des applications à courte portée

1.3.3. Topologie

L'établissement de la bande passante de communication et du protocole de fonctionnement des objets connectés dépend de la structure de déploiement des objets connectés pour les applications d'agriculture intelligente. Les structures réseau pour l'agriculture intelligente comportent généralement deux principaux types de nœuds : les nœuds de capteurs et les nœuds de liaison.

Le tableau 1.1 montre des exemples sur les technologies de communication :

Type	Spectrum	Transmission Distance	Type of Network	Frequency	Data Rate
802.11a/b/g/n/ac	Unlicensed	100 m	WLAN	2.4–5 GHz	2–700 Mbps
802.11ah	Unlicensed	1000 m	WLAN	Several Sub-GHz	78 Mbps
802.11p	Licensed	1 km	WLAN	5.9 GHz	3–27 Mbps
802.11af	Licensed	1 km	WLAN	54–790	25–550 Mbps
SigFox	Licensed	Rural: 50 km Urban:10 km	LPWA	Zwave	100–600 bps
LoRaWAN	Licensed	20 km	LPWA	Several Sub-GHz	0.3–100 kbps
NB-IoT	Licensed	35 km	LPWA	Zwave	250 kbps
LTE-3GPP	Licensed	5 km	WWAN	1.4 MHz	200 kbps
EC-GPRS	Licensed	5 m	WWAN	GSM bands	240 kbps
WiMAX	Hybrid	50–80 km	WWAN	Several Sub-GHz	70 Mbps
Bluetooth	Unlicensed	100 m	WPAN	2.4 GHz	2–26 Mbps
ZigBee	Unlicensed	1 km	WHAN	2.4 GHz	250 kbps

Tableau 1.1 : les technologies de communication typiques pour l’agriculture intelligente.

1.4. Sources d’énergie dans les systèmes IOT

Dans les systèmes IoT le choix et la gestion des sources d’énergie sont essentiels pour assurer un fonctionnement fiable, autonome et durable, surtout lorsque les dispositifs sont déployés dans des environnements isolés ou difficiles d’accès.

1.4.1. Batteries rechargeable

Les batteries Li-ion (Lithium-Ion) et Li-Po (Lithium-Polymère) sont parmi les plus couramment utilisées dans les systèmes IoT portables ou autonomes. Elles offrent une forte densité énergétique, une durée de vie relativement longue et sont rechargeables des centaines de fois. Leur tension stable (souvent 3.7V) convient parfaitement aux modules comme l'ESP32.

1.4.2. Panneaux solaires (Énergie solaire)

L'énergie solaire est une solution idéale pour les dispositifs IoT déployés en extérieur, comme dans l'agriculture intelligente, les stations météo ou les systèmes de surveillance à distance. Les panneaux solaires sont souvent associés à une batterie pour stocker l'énergie et à un contrôleur de charge pour gérer l'alimentation.

1.4.3. Alimentation secteur (AC-DC)

Dans les environnements urbains ou industriels, les dispositifs IoT peuvent être alimentés directement via une source secteur (220V AC) à travers des convertisseurs AC-DC qui délivrent des tensions de 5V ou 3.3V.

1.4.5. Alimentation USB

Pour les tests, les démos ou les modules IoT à faible consommation, l'alimentation via USB (5V) est souvent utilisée. Elle permet une alimentation directe depuis un ordinateur, une batterie externe (powerbank), ou un adaptateur mural USB. Cela est pratique, mais limité en autonomie si l'alimentation ne provient pas d'une source fixe.

1.5. Analyse et stockage des données avec un serveur (back-end)

Le serveur back-end agit comme le cerveau du système, il centralise la logique, assure la sécurité des données, maintient l'intégrité des échanges, et veille au bon déroulement des opérations. Dans les systèmes connectés, notamment ceux impliquant des objets intelligents ou des capteurs, le rôle du back-end devient encore plus crucial car il assure une interface fiable entre les dispositifs physiques et les applications logicielles. Ainsi, la compréhension de l'architecture back-end est indispensable pour saisir les mécanismes internes d'un système informatique, et pour pouvoir le concevoir, le maintenir, et l'optimiser efficacement. Dans ce qui suit, nous allons explorer plus en détail

l'infrastructure back-end mise en place dans le cadre de notre projet, en mettant en évidence ses composantes principales, son rôle dans la communication entre les différents éléments du système, ainsi que les choix technologiques qui ont guidé sa conception pour analyser et stocker les données, Figure1.5.

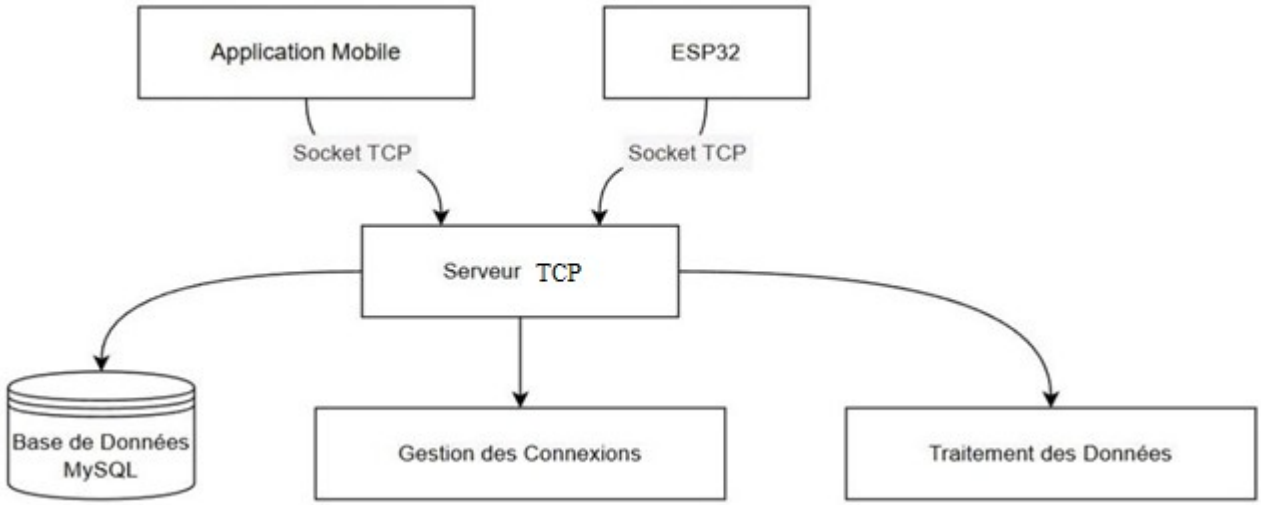


Figure 1.5 : Schéma logique du serveur

1.6. Les capteurs

Les capteurs sont spécifiquement conçus pour fonctionner dans des environnements ouverts et fermés, dans le sol, l'eau et l'air pour mesurer et collecter les paramètres environnementaux qui affectent la production, tels que les nutriments du sol, l'humidité, la température, etc.

1.6.1. Caractéristiques des capteurs

Dans le système IOT, plusieurs capteurs sont utilisés pour surveiller différentes variables environnementales essentielles à l'agriculture sous serre. Le tableau1.2 suivant présente les principales caractéristiques techniques de chaque capteur :

Capteur	Paramètre Mesuré	Plage de mesure	Tension d'alimentation	Sortie
DHT11	Température et humidité	Température :0°C à 50°C Humidité : 90%	3.3V – 5V	Numérique
BH1750	Intensité lumineuse	1 à 65 535 lux	3.3V à 5V	Numérique
YL-69	Humidité du sol	0 (sol sec) à 4095	3.3V – 5V	Analogique / Numérique
DS18B20	Température	-55°C à +125°C	3.0V à 5.5V	Numérique

Tableau 1.2: Principales caractéristiques techniques de chaque capteur.

Pour sélectionner les bons capteurs pour un système IoT agricole, plusieurs critères doivent être pris en compte. Il est essentiel de choisir des capteurs capables de mesurer avec précision les paramètres nécessaires tels que la température, l'humidité, la lumière ou l'humidité du sol, en fonction des besoins de la culture.

1.6.2. Montage des capteurs sur un Microcontrôleur

Les capteurs sont des dispositifs capables de détecter une quantité physique, chimique ou biologique et de la convertir en un signal électrique utilisable. Chaque capteur possède son propre montage.

1.6.2.1. Montage du capteur DHT11 (Température et humidité de l'air)

Le capteur DHT11 est un capteur numérique qui permet de mesurer la température ambiante ainsi que le taux d'humidité relative. Il est largement utilisé dans les systèmes de surveillance climatique pour les serres agricoles. Le montage consiste à connecter la broche DATA du capteur à une entrée numérique de Microcontrôleur, Figure1.6.

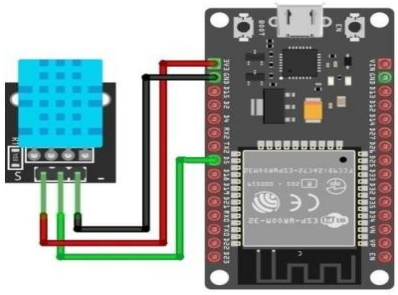


Figure 1.6**:** montage du capteur dht11 sur un Microcontrôleur

1.6.2.2. Montage du capteur YL-69 (Humidité du sol)

Le capteur YL-69 est utilisé pour mesurer le niveau d'humidité dans le sol. Il fonctionne comme une sonde résistive dont la conductivité change selon la teneur en eau du sol. Ce capteur est utile pour automatiser l'irrigation dans une serre. Le montage comprend deux parties, la sonde (insérée dans le sol) et le module électronique, dont la sortie analogique est reliée à une broche analogique de Microcontrôleur, Figure1.7.

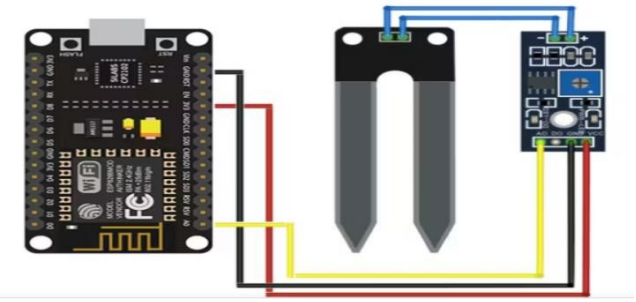


Figure 1.7 : montage du capteur YL69 sur un Microcontrôleur.

1.6.2.3. Montage du capteur BH1750 (Luminosité)

Le capteur BH1750 est une photorésistance qui varie sa résistance en fonction de l'intensité lumineuse ambiante. Plus la lumière est forte, plus la résistance diminue, et inversement. Ce capteur est particulièrement utile dans les serres pour surveiller la quantité de lumière naturelle reçue par les plantes, Figure1.8.

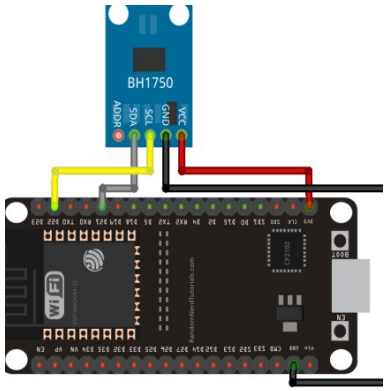


Figure 1.8**:** montage du capteur NH1750 sur un Microcontrôleur

1.6.2.4. Montage du capteur DS18B20 (Température haute précision)

Le DS18B20 est une sonde à résistance de platine très précise pour mesurer la température, notamment utile lorsque des mesures exactes sont nécessaires sur de larges plages. Elle est souvent utilisée dans les environnements industriels ou agricoles avancés, Figure1.9.

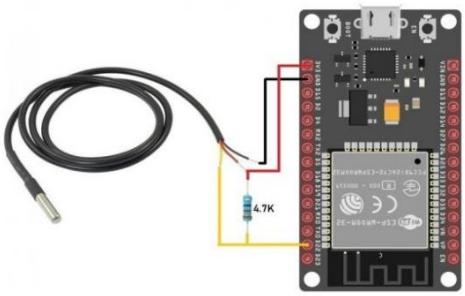


Figure 1.9**:** montage du capteur DS18B20 sur un Microcontrôleur.

1.7. Les microcontrôleurs

Un microcontrôleur est un composant électronique qui fonctionne comme un mini-ordinateur intégré dans des objets du quotidien. Il est composé principalement de trois parties :

- Un processeur : qui exécute les instructions.
- Une mémoire : pour stocker les données et le programme.
- Des entrées/sorties : pour communiquer avec d'autres composants comme les capteurs, les moteur, et autres.

1.7.1. Avantages des microcontrôleurs

Parmi les avantages des microcontrôleurs :

- Petite taille : facile à intégrer dans tout type d'objet.
- Faible coût : permet de fabriquer des objets connectés à grande échelle.
- Basse consommation d'énergie : idéal pour les objets autonomes.
- Faciles à programmer : souvent utilisés avec Arduino, ESP32, STM32, etc.

1.7.2. Le rôle des microcontrôleurs du système IoT dans l'agriculture

Les microcontrôleurs sont le cœur des objets connectés. Sans eux, il serait impossible de transformer un objet physique en un système intelligent capable d'interagir, de communiquer et de s'adapter à son environnement. Le microcontrôleur peut :

- Lire les capteurs : Il récupère les données des capteurs comme la température, l'humidité et la lumière.
- Traiter les données : Il peut analyser ou filtrer les données pour décider quoi faire ensuite.
- Contrôler des actionneurs : Par exemple, il peut allumer un ventilateur, activer une pompe.
- Communiquer avec d'autres appareils : Grâce à des modules de communication Wi-Fi.

Le tableau 1.3 montre les différences entre les microcontrôleurs les plus couramment utilisés :

Microcontrôleur	**Mémoire**	**Tension**	**Avantages pour l'agriculture**
Arduino UNO	32 KB / 2 KB	5V	Facile à utiliser mais Pas de Wi-Fi ou Bluetooth intégrés,
ESP32	4 MB / 512 KB+	3.3V	Wi-Fi et Bluetooth intégrés, puissant, idéal pour les systèmes connectés
STM32	64 KB / 20 KB	3.3V	Bonne performance, faible consommation, adapté aux projets en temps réel.

Tableau 1.3 : Tableau suivant montre les différences distinctes entre les microcontrôleurs.

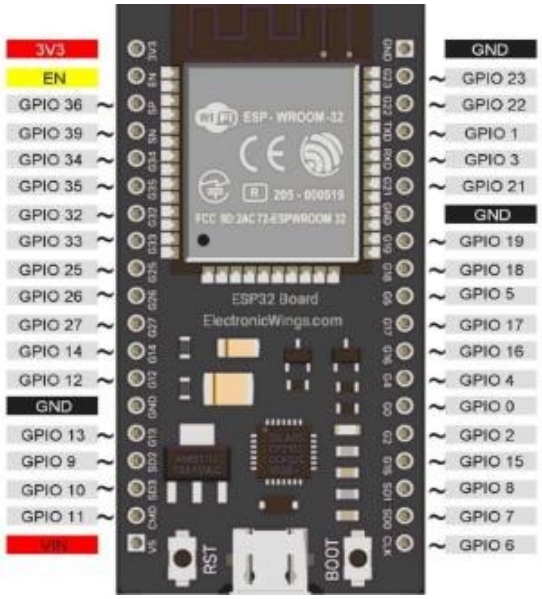
Pour un système de serre connectée, l'ESP32 est souvent le plus adapté : il intègre directement le WiFi et peut lire des capteurs tout en contrôlant des relais.

1.7.3. La carte ESP32

L'ESP32 présente une intégration élevée avec des commutateurs d'antenne intégrés, un convertisseur d'équilibre RF, un amplificateur de puissance, un amplificateur de récepteur à faible bruit, des filtres et des modules de gestion de l'alimentation, Figure 1.10. Il est le Cœur des Projets IoT agricoles grâce a aux points forts que nous montrons dans le tableau ci-dessous.

Avantage	**Description**
Wi-Fi + Bluetooth	Connexion sans fil intégrée (Wi-Fi 2.4 GHz, BT v4.2 et BLE)
Dual-core	2 cœurs cadencés jusqu'à 240 MHz (performant pour traitements rapides)
Multitude de GPIO	Jusqu'à 34 broches programmables, supportant ADC, DAC, PWM, SPI, I2C, etc.
Faible consommation	Modes Deep Sleep pour l'IoT (autonomie prolongée sur batterie)
Sécurité matérielle	Prise en charge native du chiffrement (AES, RSA, SHA-2), boot sécurisé
Compatibilité	Programmable avec Arduino IDE, MicroPython, ESP-IDF, etc.

Tableau 1.4 : Les points forts de l'ES32.



Pins d'entrée : EN, GPIO 34, 35, 36, 39
Touch (Tactile) : GPIO 32, 33, 27, 14, 12, 13, 15, 2, 0, 4
PWM (Modulation de largeur d'impulsion) : Tous les pins de sortie, sauf GPIO 34 à 39
État HAUT au démarrage (BOOT) : Pins 1, 3, 5, 14, 15 (Si une LED ou un relais est connecté à ces pins, cela peut provoquer des activations non souhaitées !)
DAC (Convertisseur numérique vers analogique) : GPIO 25, 26
ADC (Convertisseur analogique vers numérique) : GPIO 0, 2, 4, 12–15, 25–27, 32–39
À ne pas utiliser : GPIO 6 à 11 (Ces pins sont connectés en

Figure 1.10 : Description des entrées/sorties de la carte ESP32.

1.8. Applications mobiles

Les applications mobiles sont devenues des outils incontournables dans la société moderne, s'intégrant dans presque tous les secteurs économiques et industriels. Leur principal avantage réside dans leur accessibilité : elles permettent aux utilisateurs de consulter des informations et de gérer diverses tâches en tout lieu et à tout moment. Grâce à des interfaces intuitives et conviviales, elles facilitent l'interaction entre l'utilisateur et des systèmes numériques parfois complexes.

Dans un système IoT, les capteurs collectent des données dans leur environnement et les transmettent à une plate-forme de traitement via des réseaux sans fil.

Une application mobile joue alors un rôle central en tant qu'interface utilisateur permettant de visualiser ces données en temps réel et d'assurer la commande des actionneurs à distance.

1.9. Site web

Le site web est un ensemble de pages accessibles via Internet, regroupées sous un même nom de domaine. Il peut être consulté à partir d'un navigateur web (comme Chrome, Firefox, Safari…) sur un ordinateur, une tablette ou un smartphone.

1.9.1. Types des sites Web

- Site vitrine : Présente une entreprise, ses services, ses contacts.
- Site e-commerce : Vente de produits en ligne.
- Application Web : Outils interactifs en ligne.

1.9.2. Site Vitrine pour un Système IoT

Un site vitrine IoT est un site web destiné à présenter et promouvoir un système connecté, comme une solution IoT pour l'agriculture, la domotique, l'industrie. Il ne permet pas nécessairement d'interagir directement avec le système en temps réel, mais sert à informer, valoriser, et rassurer les visiteurs.

1.9.3. Objectifs du Site Vitrine pour les IoT

- Présenter le système IoT.
- Mettre en avant ses avantages.
- Gagner la confiance des clients ou partenaires.
- Donner un moyen de contact.

Un site vitrine pour un système IoT est comme une carte de visite numérique. il présente le projet, rassure les visiteurs, attire des des clients, et explique le potentiel de la technologie.

1.10. Conclusion

Grâce à l'Internet des Objets (IoT), les agriculteurs peuvent mieux répondre aux différents besoins en utilisant des appareils connectés et des données en temps réel. Aujourd'hui, des millions d'appareils IoT sont déjà utilisés dans l'agriculture, et ce nombre continue d'augmenter chaque jour. Pour accompagner cette évolution, il faut développer une application mobile et un site web vitrine qui permettent de présenter, surveiller et valoriser les systèmes connectées pour les serres. Ces outils facilitent l'accès à l'information, la gestion des capteurs, et offrent une meilleure visibilité de notre solution.

2. Chapitre 2 : Développement du serveur et du code de l’ESP32

2.1. Introduction

Dans le cadre de ce projet, nous avons développé un serveur fonctionnant sous le système Debian (Linux) , visant à fournir un canal de communication sécurisé entre le ESP32 et l'application (Android) pour l'envoi des commandes et la réception des données. Lors du choix de l'environnement d'exploitation et du langage de programmation, un certain nombre de facteurs ont été pris en compte, dont les plus importants sont : la stabilité, les performances, la facilité de maintenance et la sécurité.

Le choix de Debian comme système d'exploitation offre une stabilité à long terme, une faible consommation de ressources, est open source et largement pris en charge par une large communauté, cela facilite la gestion et la mise à jour régulière et sécurisée du serveur, figure 2.1.

Install Type	RAM (minimum)	RAM (recommended)	Hard Drive
No desktop	256 megabytes	512 megabytes	4 gigabytes
With Desktop	1 gigabytes	2 gigabytes	10 gigabytes

Tableau 2.1 : Debian configuration requise.

En termes de programmation, le langage Python a été choisi en raison de sa simplicité et de sa clarté, permettant un développement de serveur rapide et efficace, avec des modifications ou des extensions ultérieures facilement possibles sans avoir besoin d'une restructuration complète. Python propose également un grand nombre de bibliothèques prêtes à l'emploi pour gérer les réseaux et les bases de données, ce qui le rend très adapté à ce type de serveur léger et de taille moyenne.

Le protocole de communication TCP (Transmission Control Protocole) a été choisi car il offre une grande fiabilité dans la transmission des données. Contrairement à d'autres protocoles tels que UDP, TCP garantit que les données parviennent à l'autre extrémité dans un ordre complet et ordonné, grâce à un mécanisme de vérification des erreurs et de retransmission des paquets perdus. Cela le rend adapté aux applications qui nécessitent une connectivité stable et une transmission d'informations précise, comme la transmission de lectures de capteurs ou de commandes de contrôle.

Figure 2.1 : La relation entre les composants (Python, ESP32, Application)

2.2. Structure du serveur :

Pour garder le code organisé et faciliter la maintenance et l'extension futures, nous avons divisé le projet en plusieurs fichiers, chacun avec un rôle spécifique et clair au sein de l'architecture du serveur.

Cette méthode permet de :

- Faciliter la compréhension lors du développement ou lors de la transmission du projet à quelqu'un d'autre.
- La possibilité de modifier une partie spécifique du serveur sans affecter le reste des parties.
- Améliorer la réutilisabilité et l'extensibilité.

Étant donné que le serveur a plusieurs tâches différentes (par exemple, écouter les connexions, gérer les utilisateurs connectés, communiquer avec la base de données...), alors diviser le projet en modules est un choix logique et efficace, figure 2.3.

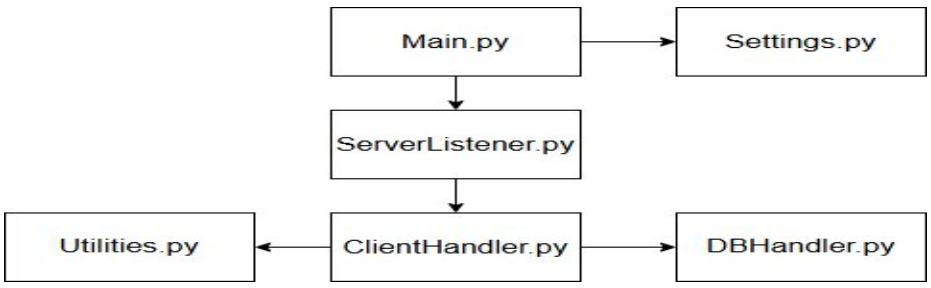


Figure 2.2 : Schéma de l'architecture du serveur et relations entre les fichiers.

Le tableau ci-dessous montre un aperçu structuré des principaux fichiers du serveur, avec une brève description de leur rôle dans la structure du projet :

Fichier	Nom	Role
Main.py	Fichier principal	Responsable de l'exploitation du serveur et de la configuration des composants de base.
ServerListener.py	Reçoit des connexions	Contient du code permettant d'écouter les connexions TCP entrants.
ClientHandler.py	Classe client	Représente Chaque appareil connecté (ESP ou application)
DBHandler.py	Gestion de base de données	Exécute des requêtes sur la base de données MySQL (ajout de données, requêtes, vérification...).
Settings.py	Paramètres principaux	Contient les paramètres réseau (IP, port) et les données de connexion à la base de données.
Utilities.py	Nettoyeur de chaînes d'octets	Traitement et conversion de données d'octets en texte
GreenApi.py	Assistant d'agriculture intelligente pour application	Fournit des images de plantes pour l'application. représente un point intermédiaire entre l'utilisateur et le système intelligent.

Tableau 2.*2* : Répartition des fichiers du serveur et leurs rôles.

2.2.1. Gestion de la base de données (DBHandler.py) :

Le fichier DBHandler.py joue un rôle central dans la gestion des interactions entre le serveur Python et la base de données MySQL. Il inclut toutes les opérations de lecture et d'écriture, permettant ainsi une séparation claire entre la logique applicative et la couche de persistance des données.

Ce module est responsable notamment de :

- L'authentification des utilisateurs (vérification du nom d'utilisateur et du mot de passe).
- La gestion des appareils connectés (ESP32) et des données associées (Temp, …).
- L'organisation des groupes d'appareils par pièces ou catégories.
- Le stockage et la récupération des mesures envoyées par les capteurs.

Le fichier utilise la bibliothèque mysql.connector pour établir une connexion sécurisée à la base de données et implémente une logique centralisée afin que toutes les requêtes SQL soient traitées au même endroit, facilitant ainsi la maintenance et réduisant le risque d'erreurs.

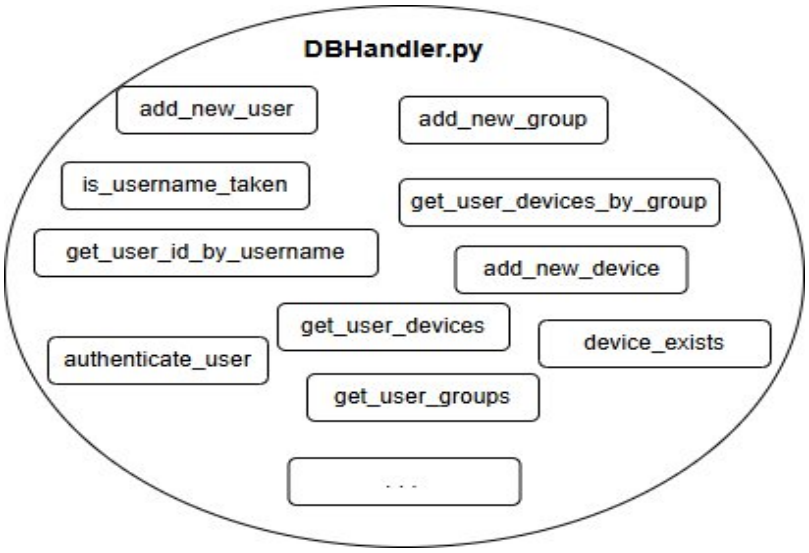


Figure 2.3 : Les fonctions du fichier DBHandler.py.

2.2.2. Configuration du serveur – Settings.py :

Ce fichier contient les paramètres de configuration de base du serveur. Il définit notamment les informations de connexion à la base de données (utilisateur, mot de passe, nom de l'hôte et de la base de données) ainsi que les paramètres réseau du serveur (adresse IP et port d'écoute).En outre, une structure est en place pour gérer les clients connectés de manière dynamique. Une liste globale est utilisée pour enregistrer les connexions actives, ce qui permet au serveur de savoir quels appareils ou utilisateurs sont actuellement connectés. Une fonction d'initialisation est fournie pour vider et réinitialiser cette liste si le serveur est redémarré ou réinitialisé.

```
DB_CONFIG = {
    'user': "root",
    'password': "Passowrd@123",
    'host': 'localhost',
    'database': 'users'
}

SERVER_CONGIG = {
    'ip'   : '192.168.1.50',
    'port' : 57005,
}
```

Figure 2.4 : Paramètres du serveur dans le fichier Settings.py.

2.3. Serveur principal – ServerListener.py :

Ce fichier contient la classe principale responsable de la gestion du serveur TCP. Son rôle est d'écouter et d'accepter les connexions entrantes des clients et de déléguer le traitement à une fonction spécifique (appelée handler).

Lors de l'initialisation, l'adresse IP du serveur et le port sont définis. Le serveur démarre alors un thread dédié à l'écoute des connexions. Cette séparation permet d'écouter en parallèle sans perturber l'application principale.Le serveur utilise une boucle continue pour accepter les connexions, et chaque client connecté est traité individuellement via la fonction passée en paramètre. En cas d'erreur ou d'arrêt du serveur, les ressources sont libérées proprement (fermeture du socket et arrêt du thread).

Ce mécanisme garantit un fonctionnement stable, réactif et modulaire, ce qui facilite la maintenance et l'adaptation à différents types de clients.

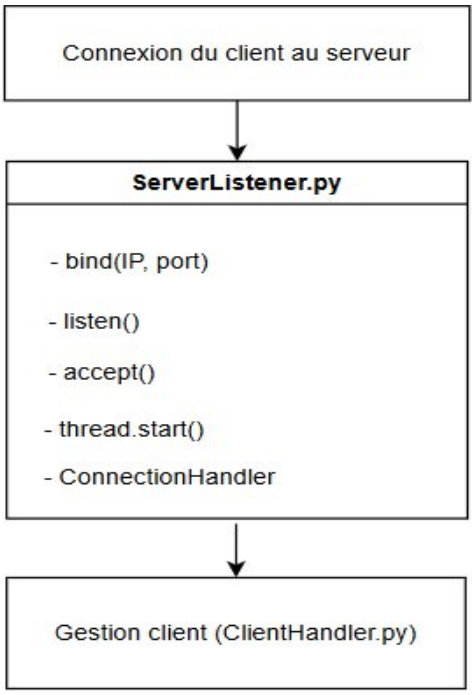


Figure 2.5 : Schéma de fonctionnement du serveur TCP (ServerListener.py).

2.3.1. Gestion d'un client connecté – ClientHandler.py :

La classe Client représente un client connecté au serveur via une socket TCP. Chaque instance gère un utilisateur (ou un ESP32) et exécute les protocoles nécessaires pour l'authentification, la gestion des groupes/appareils et la transmission des données en temps réel.

Les variables internes importantes d'un client :

- Username, Password, UserID : informations de l'utilisateur.
- ESP32Code : identifiant de l'appareil connecté (ESP32).
- Groups, Devices : listes des groupes et appareils liés à l'utilisateur.
- deviceType : 1 = ESP32, 2 = application mobile.
- IsLogin, IsConnected : états de connexion.

2.3.2. Réception d'un paquet et extraction du op_code (code d'operation) :

Lorsqu'un client envoie un message au serveur via une socket TCP, celui-ci arrive sous forme d'un paquet binaire structuré. Le serveur commence par lire 1 octet, qui indiquent la taille totale du paquet à venir. Ensuite, il lit le paquet entier selon cette taille.

Une fois le paquet complet reçu, le serveur extrait les premières données utiles grâce à une structure donnée, figure 2.7. Le premier champ de chaque paquet est toujours un entier (short) représentant le op_code, c'est-à-dire le code de l'opération à exécuter (connexion, ajout d'un appareil, envoi de données…).

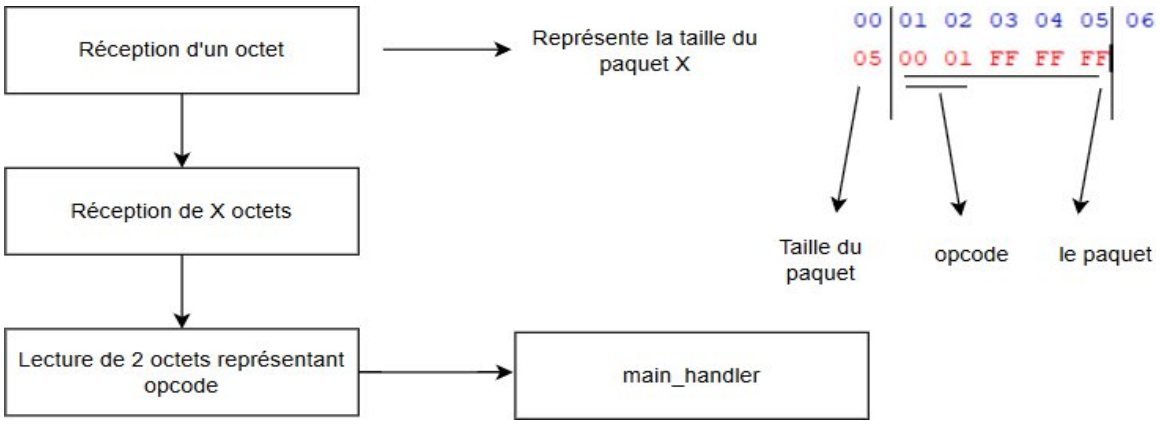


Figure 2.6 : Schéma de la réception des paquet et extraction de l'op_code.

Ce op_code est utilisé pour déterminer l'action à effectuer via un gestionnaire central (main_handler), qui redirige vers la fonction appropriée.

```python
def main_handler(self, packet: bytes):

    op_code ,= struct.unpack_from("<H", packet)
    print(f"Received opcode {op_code}")
    packet = packet[2:]

    if op_code == 1 and self.deviceType == 0 and not self.IsLogin:
        self.handle_login(packet)

    elif op_code == 3 and self.deviceType == 2 and self.IsLogin:
        self.handle_add_device(packet)
```

Figure 2.7 : Réception et traitement des paquets selon op_code

Le tableau suivant présente les différentes instructions (opcodes) utilisées dans la communication entre les clients (application mobile ou ESP32) et le serveur. Chaque ligne décrit la fonction associée à un code opérationnel, le type de client qui l'utilise, ainsi que son rôle dans le système.

Op_code	Fonction	Type de Client	role
1	handle_login	Application mobile	Authentification de l'utilisateur
3	handle_add_device	Application mobile	Ajouter un nouveau appareil
7	handle_esp32_login	ESP32	Authentification de l'ESP32
9	handle_proxy_app_to_device	Application mobile	Envoyer des informations de l'application à esp32
11	handle_proxy_device_to_app	ESP32	Transfert de données des capteurs vers l'application
13	handle_register_new_account	Application mobile	Création d'un nouveau compte utilisateur
55	handle_get_groups	Application mobile	Demander la liste des groupes d'utilisateur
60	handle_delete_device	Application mobile	Supprimer un appareil existant
62	handle_add_new_group	Application mobile	Ajouter un nouveau groupe
65	handle_delete_group	Application mobile	Supprimer un groupe existant
70	handle_get_data	Application mobile	Récupérer les données historiques des capteurs

Tableau 2.*3* : Répartition des fichiers du serveur et leurs rôles.

2.4. Présentation du fichier serveur secondaire (GreenApi.py) :

Ce module secondaire développé avec Flask joue un rôle complémentaire essentiel dans le projet. Il fournit deux fonctionnalités principales :

- Distribution des icônes des plantes utilisées dans l'application mobile.
- Service de réponse intelligente (chatbot) basé sur l'API Gemini de Google pour répondre aux questions des utilisateurs sur le projet.

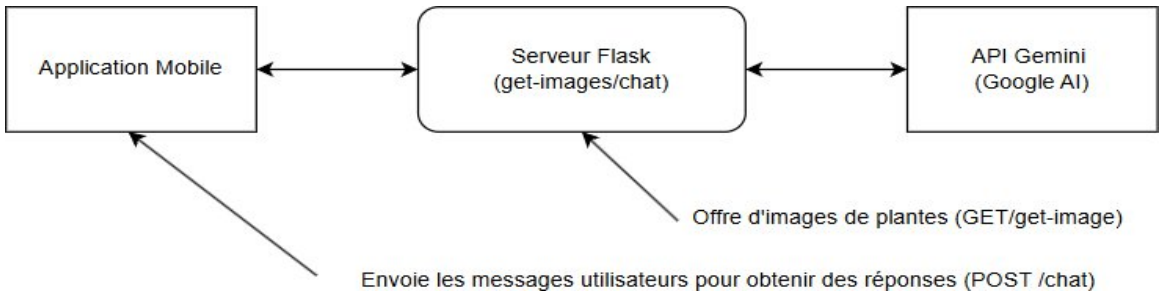


Figure 2.8 : Schéma fonctionnel du serveur Flask.

2.5. Installation et déploiement du serveur sous Debian :

2.5.1. Préparation de l'environnement :

Avant de procéder à l'installation du serveur, il est important de s'assurer que le système Debian est à jour :

```
root@srv824330:~# sudo apt update
Hit:1 http://security.debian.org/debian-security bullseye-security InRelease
Hit:2 http://deb.debian.org/debian bullseye InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Hit:4 http://deb.debian.org/debian bullseye-backports InRelease
Hit:5 https://repository.monarx.com/repository/debian-bullseye bullseye InReleas
e
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@srv824330:~# sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Figure 2.9 : mise à jour du système Debian.

Ensuite, il faut installer Python3 et le gestionnaire de paquets pip si ce n'est pas déjà fait :

sudo apt install python3 python3-pip

- Installation des dépendances :

Nous installons les bibliothèques nécessaires à l'aide de la commande suivante :

Pip3 install mysql.connector requests Flask

- Lancement du serveur : Pour démarrer le serveur, il suffit d'exécuter le fichier principal en Python

Python3 Main.py

2.6. Fonctionnement du code de l'ESP32 :

Le code embarqué développé en C++ pour le ESP32 permet la lecture périodique de plusieurs capteurs et la communication avec le serveur via le protocole TCP (GreenServer). Le ESP32 est équipé des capteurs suivants :

- **DHT11** : température et humidité de l'air (pin 21).
- **BH1750** : luminosité ambiante via I2C (SDA 21, SCL 22).
- **DS18B20** : température de sol (pin 19).

- **YL-69** : Capteur d'humidité du sol (pin 36).
- **Relais** : contrôle de pompe, Ventilateur, La lumière, Chauffage (pin 25, 26 ,27 ,32).

Lors du démarrage, le programme vérifie si les identifiants WiFi sont enregistrés dans la mémoire EEPROM. Si oui, il se connecte automatiquement au réseau et tente d'établir une connexion TCP avec le serveur. En cas d'absence de configuration Wifi, l'ESP32 crée un point d'accès (AP) pour permettre à l'utilisateur d'envoyer les identifiants (SSID, PASSWORD) via l'application mobile. Ces identifiants sont ensuite sauvegardés pour les connexions futures. Une fois connecté au serveur, le ESP32 utilise une classe personnalisée GreenESP pour envoyer et recevoir des paquets de données. Chaque paquet suit une structure définie contenant un code d'opération (opcode), un type de donnée (tableau 2.3), et une valeur (float ou int). Le ESP32 envoie toutes les **5 secondes** les valeurs lues des capteurs avec l'opcode 11, tandis qu'il peut aussi recevoir des commandes (par exemple, inversion de l'état du relais avec l'opcode 1).

Type	**Valeur**
1	Température du sol
2	Humidité du sol
3	Température de l'air
4	Humidité de l'air
5	Niveau de lumière

Tableau 2.*4* : Table de correspondance des types de valeurs mesurées par les capteurs.

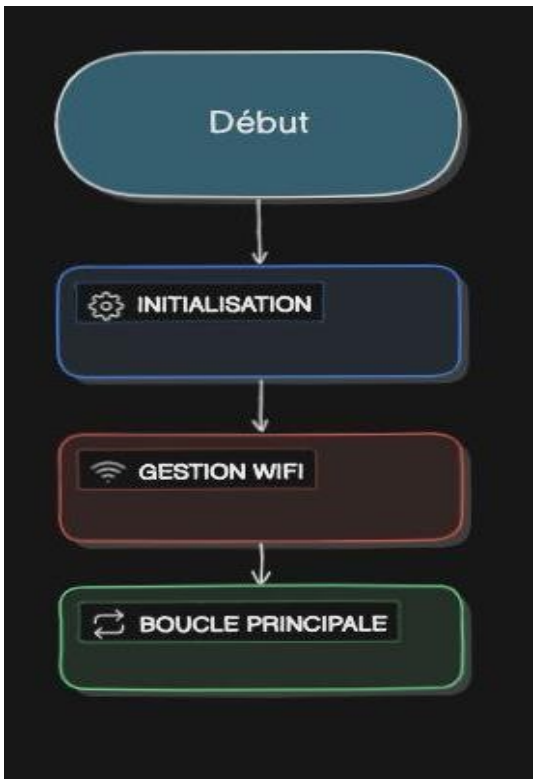


Figure 2.10 : Diagramme de Fonctionnement du code de ESP32.

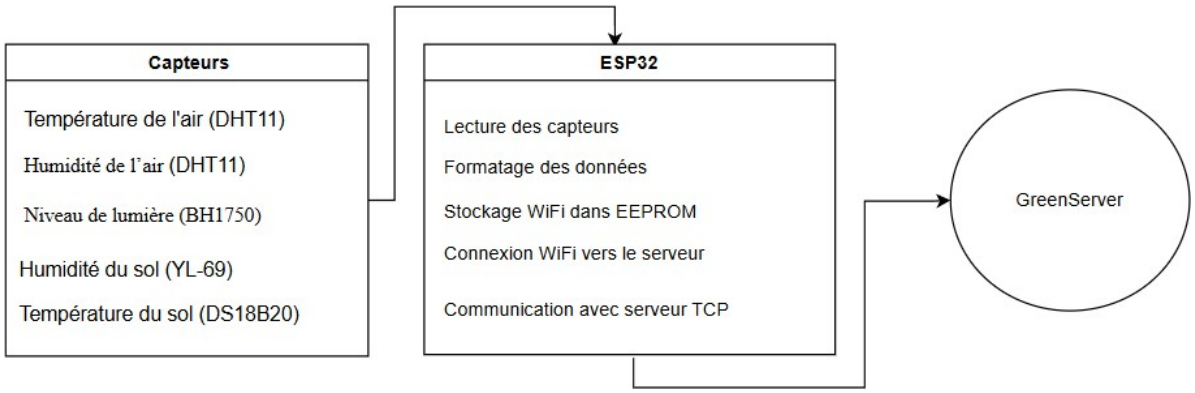
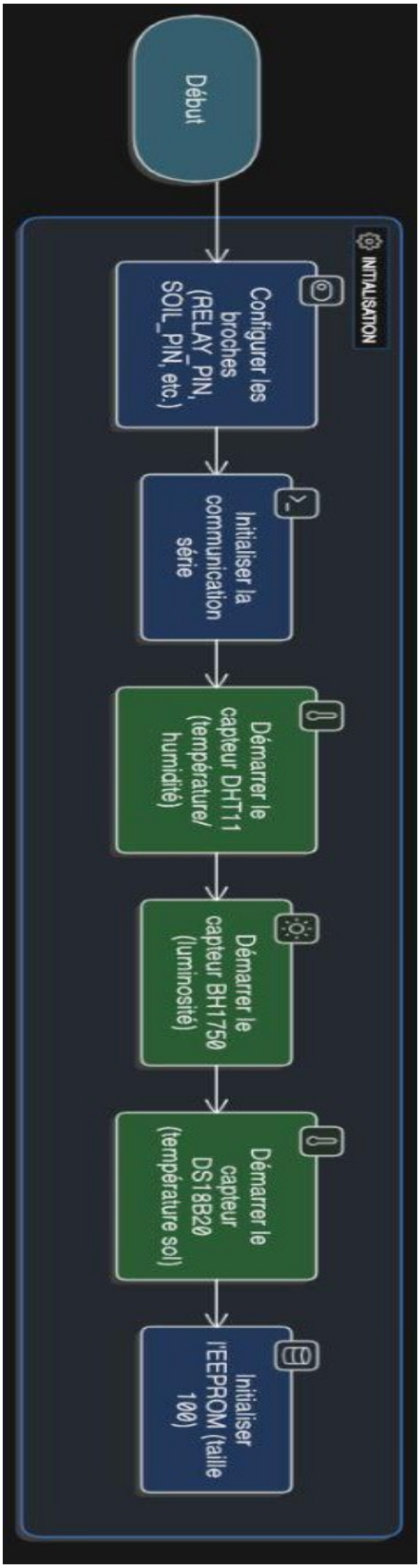
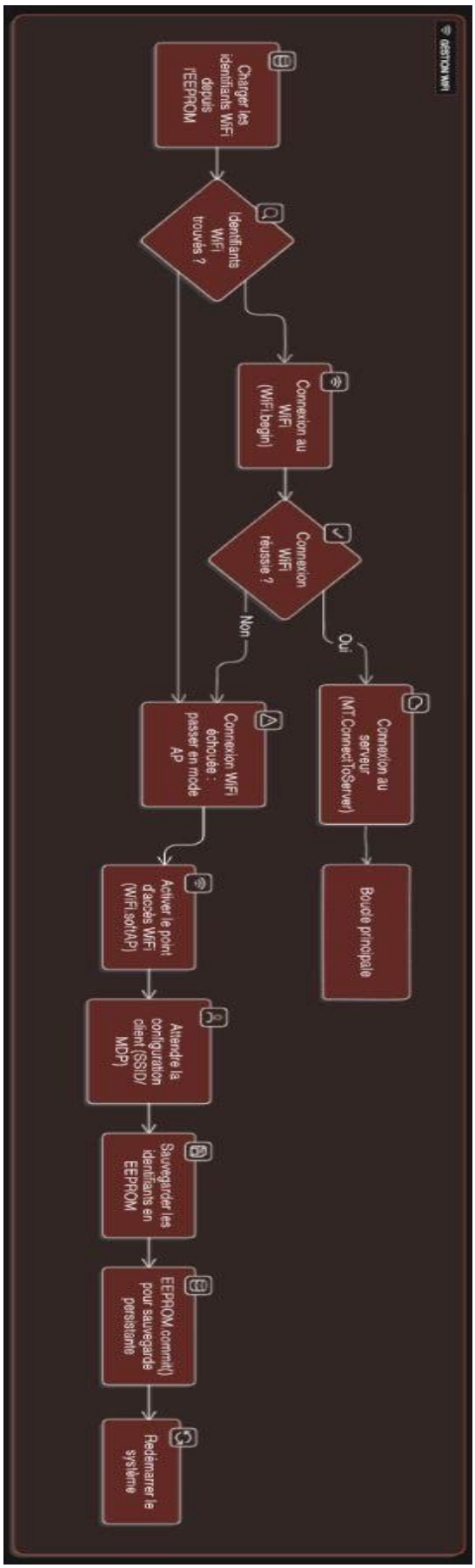
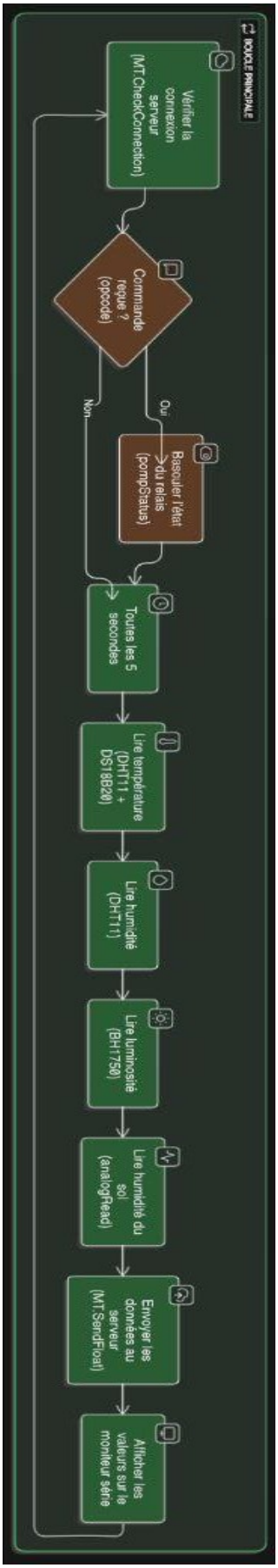


Figure 2.1 : Schéma fonctionnel de l'ESP32.


Début
INITIALISATION
Configurer les broches (RELAY_PIN, SOIL_PIN, etc.)
Initialiser la communication série
Démarrer le capteur DHT11 (température/ humidité)
Démarrer le capteur BH1750 (luminosité)
Démarrer le capteur DS18B20 (température sol)
Initialiser l'EEPROM (taille 100)


GESTION WIFI
Charger les identifiants WiFi depuis l'EEPROM
Identifiants WiFi trouvés ?
Connexion au WiFi (WiFi.begin)
Connexion WiFi réussie ?
Non
Oui
Connexion au serveur (MT.connectToServer)
Boucle principale
Connexion WiFi échouée : passer en mode AP
Activer le point d'accès WiFi (WiFi.softAP)
Attendre la configuration client (SSID/MDP)
Sauvegarder les identifiants en EEPROM
EEPROM.commit() pour sauvegarde persistante
Redémarrer le système


BOUCLE PRINCIPALE
Vérifier la connexion serveur (MT.CheckConnection)
Commande reçue ? (opcode)
Oui
Non
Basculer l'état du relais (pompStatus)
Toutes les 5 secondes
Lire température (DHT11 + DS18B20)
Lire humidité (DHT11)
Lire luminosité (BH1750)
Lire humidité du sol (analogRead)
Envoyer les données au serveur (MT.SendFloat)
Afficher les valeurs sur le moniteur série

2.6.1. Diagramme schématique :

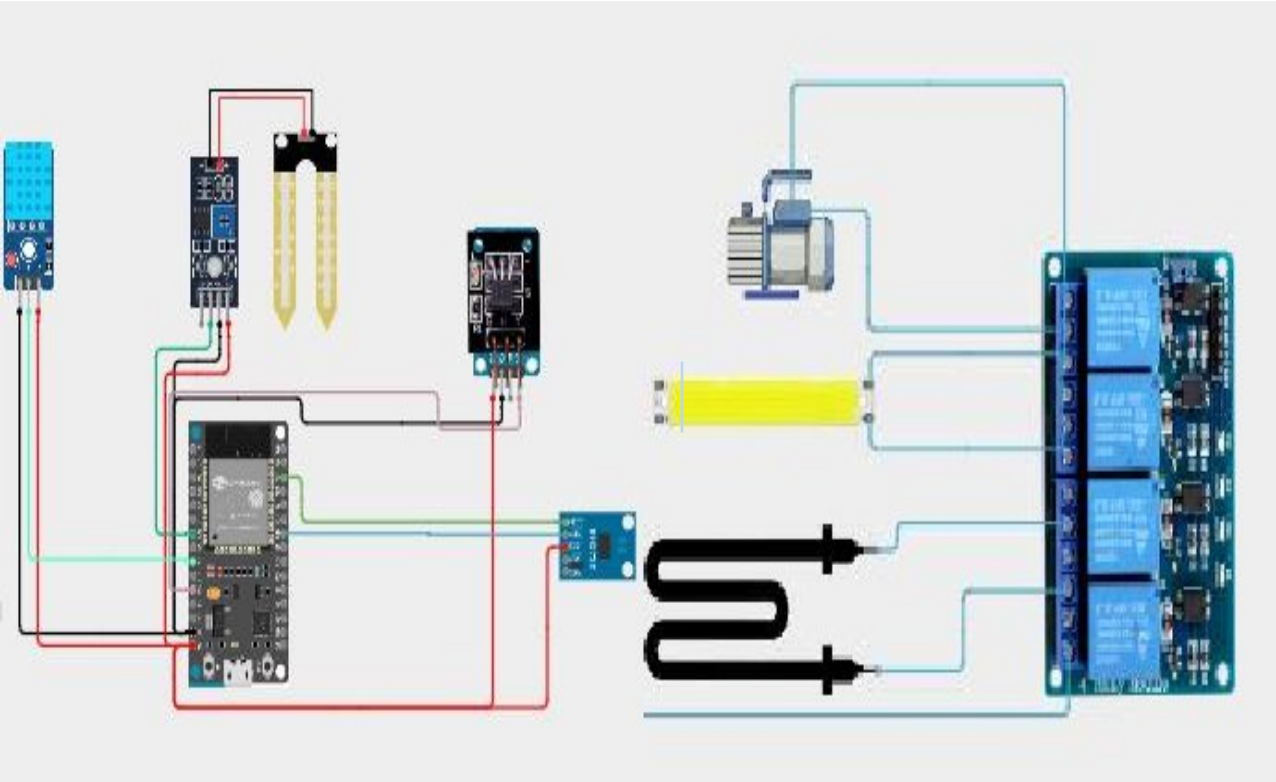


Figure 2.2 : Diagramme schématique de système.

2.7. Conclusion :

Dans ce chapitre, nous avons présenté en détail l'architecture logicielle du serveur et le fonctionnement du code embarqué de l'ESP32. Grâce à un serveur structuré en modules sous Debian utilisant Python et le protocole TCP, nous avons pu établir une communication fiable et sécurisée entre les différents composants du système.

3. Chapitre 3 : Développement de l'Application Android et le site web

3.1. Introduction

Ce chapitre traite du développement des deux interfaces principales du système IoT : une application mobile Android, qui permet aux utilisateurs de surveiller les capteurs et de contrôler les équipements à distance, et un site web vitrine**,** destiné à présenter le projet et faciliter la communication avec les clients.

L'application a été développée avec Android Studio en utilisant des fragments modulaires, tandis que le site web repose sur les technologies React.js et Node.js. Ces interfaces assurent l'accessibilité, l'interaction et la valorisation du système de serre intelligente.

3.2. L'application mobile Green house :

L'application Android développée constitue un outil de supervision et de contrôle à distance d'une serre connectée. Elle permet à l'utilisateur de visualiser en temps réel les données transmises par les capteurs installés (température, humidité, luminosité…), et d'interagir avec les actionneurs (pompe, ventilation, chauffage) via des commandes envoyées au serveur. Grâce à son interface moderne, conçue sous Android Studio, l'application assure une expérience utilisateur fluide, combinant la visualisation graphique des mesures, la navigation entre différentes pièces et le contrôle direct des équipements.

Afin de mieux comprendre la logique de fonctionnement de cette application mobile, la section suivante présente l'architecture logicielle de l'application ainsi que l'organisation de ses principaux composants sur android studio, qui est un outil officiel fourni par intel pour créer et développer des applications Android. Il facilite toutes les étapes du développement, de la conception de l'interface utilisateur (UI) à la programmation. Avec une architecture principale :

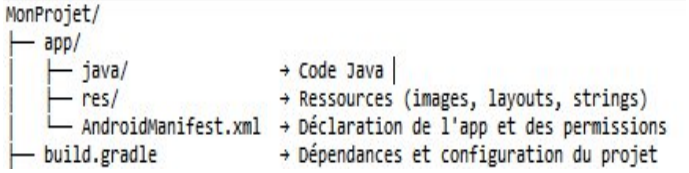


Figure 3.1 : l'architecture logicielle de l'application.

3.2.1. Architecture de l'application green house :

L'architecture de l'application Android développée pour la gestion d'une serre connectée repose sur une structure modulaire et orientée utilisateur. Elle est conçue pour offrir une interface intuitive tout en permettant une communication fluide avec les composants IoT. La figure ci-dessous illustre l'architecture fonctionnelle de l'application Android. Celle-ci repose sur plusieurs fragments, chacun dédié à une fonctionnalité spécifique.

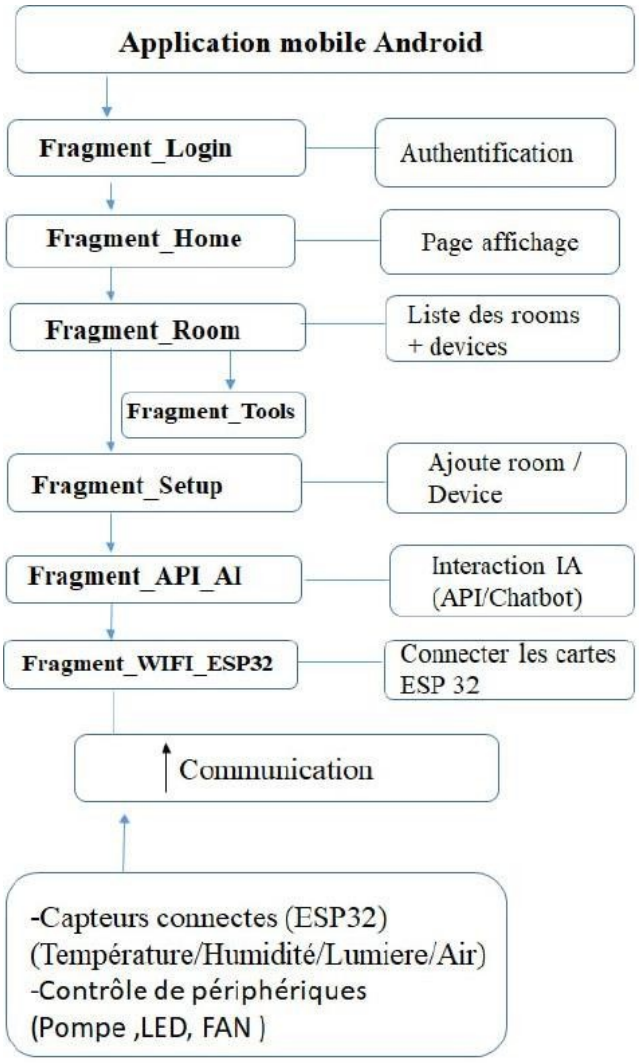


Figure 3.2 : L'architecture de l'application green house

3.3. Composition de l'UI avec le XML :

L'UI est tout ce que l'utilisateur voit et touche dans l'application. C'est l'écran, les boutons, le texte, les images, les menus, etc. Fondamentalement, c'est le look et feel de l'application.

Les éléments de l'interface utilisateur sont des composants visuels statiques et l'état (state) de l'interface utilisateur est la donnée dynamique qui affecte l'affichage, combinant les éléments et leur état pour former l'interface interactive., figure 3.3.

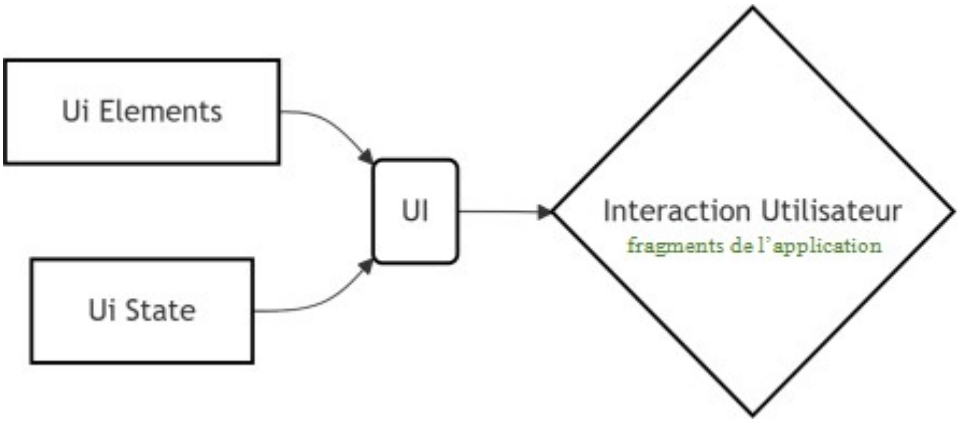


Figure 3.3 : le résultat de L'UI d'éléments de liaison liés à l'état de l'interface.

Le XML ne contient aucune information sur la façon de l'afficher. Les mêmes données XML peuvent être utilisées dans de nombreux scénarios de rendu différents. Il s'agit d'un langage de structuration des données, et non d'un langage de présentation. Sa force réside dans sa capacité à séparer clairement les données et l'affichage, la flexibilité d'utilisation, et les principaux avantages de cette séparation qui génère l'UI utilisateur et une maintenance simplifiées.

Les trois structures de XML :

a. **Hiérarchique** : des boîtes dans des boîtes (éléments qui contiennent d'autres éléments). On l'utilise quand on a beaucoup d'informations à organiser. C'est plus clair si on veut ajouter plusieurs détails pour une même chose.
b. **Attributs** : On met des infos rapides dans la balise. On l'utilise quand on a des petites informations simples et pas beaucoup de détails. C'est pratique pour des valeurs rapides.
c. **Mixte** : on utilise les deux ensembles.

Chaque fichier XML basé sur le gradle qui constitue l'élément central du système de build, automatisant l'ensemble des tâches nécessaires pour générer, tester, empaqueter et publier l'application.

3.4. Architecture des fragments de l'application green house :

Les fragments de l'application mobile Green House repose sur une architecture claire, modulaire et bien structurée, comme illustre dans la figure3.4.

Pour chaque fragment de l'application Green House, est définie à l'aide de fichiers de mise en page XML, conformément aux bonnes pratiques du développement Android. Ces fichiers décrivent la structure visuelle et les composants UI de chaque écran, indépendamment de la logique Java.

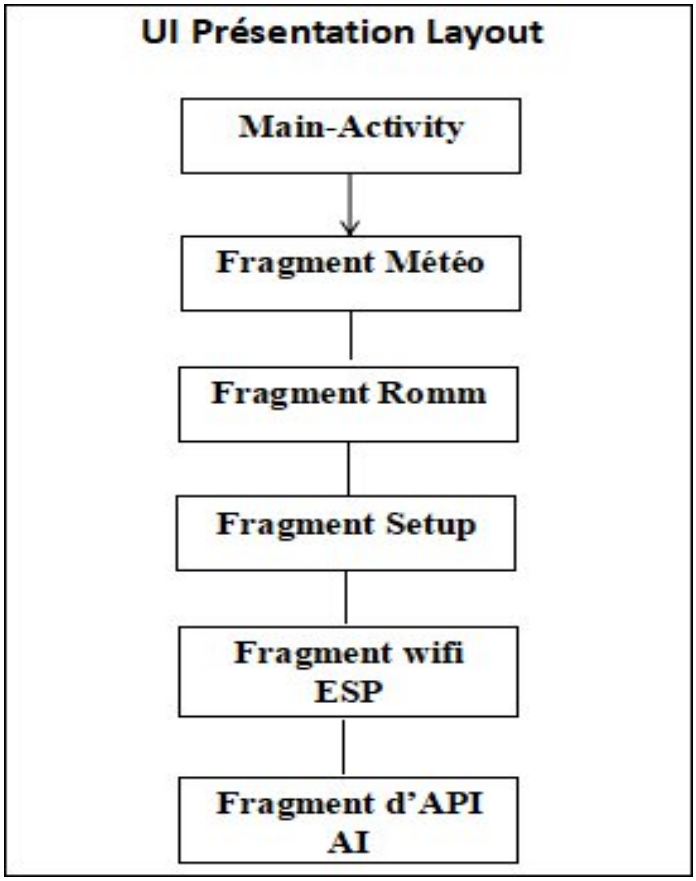


Figure 3.4 : L'architecture des fragments de l'application green house.

Cette séparation permet une meilleure lisibilité du code, une maintenance facilitée, ainsi qu'une évolutivité optimale de l'application.

3.4.1. Interfaces de fragments (Présentation Layer) :

L'interface utilisateur est un élément clé. Elle a été conçue en utilisant des fichiers XML pour offrir une expérience transparente, intuitive et accessible. L'organisation visuelle est basée sur deux couches:

3.4.1.1. Couche layout pour les interfaces :

- MainActivity agit comme un conteneur principal.
- Fragment Room est chargé de l'affichage graphique des données de chaque devices et de contrôler les actionneurs (pompe, LED, chauffage…).
- Fragment Setup permet de ajouter des Rooms et devices.
- LoginActivity permet l'accès sécurisé à l'application.
- Fragment wifi ESP permet de scanner le code QR des ESP
- Fragment API AI est chargé de l'affichage AI.

3.4.1.2. Couche Ressources

Elle regroupe tous les fichiers XML définissant l'apparence et le comportement des éléments graphiques :

- activity_main.xml, item_sensors.xml : mise en page générale et composants RecyclerView.
- strings.xml, colors.xml : gestion des traductions et des couleurs.

Le layout des interfaces essentielles dans une application Android joue un rôle crucial dans l'expérience utilisateur. L'écran de connexion permet de sécuriser l'accès à l'application en offrant une

interface claire et simple pour l'utilisateur, ce qui garantit que seules les personnes autorisées puissent accéder aux informations de la ferme. L'écran d'accueil, quant à lui, présente les fonctionnalités principales de l'application, en fournissant un aperçu rapide et intuitif des données collectées par les capteurs et des actions disponibles, facilitant ainsi la navigation. Enfin, l'écran des paramètres offre à l'utilisateur la possibilité de personnaliser l'application selon ses besoins, en ajustant des seuils ou des configurations pour améliorer l'efficacité du système, Figure3.4 .

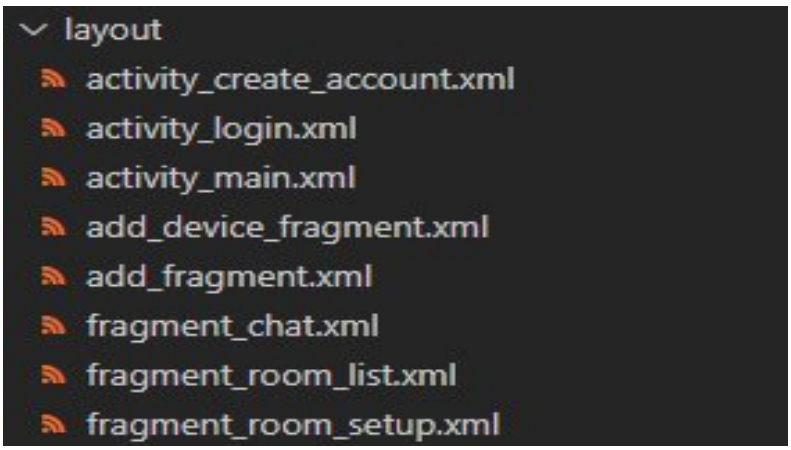


Figure 3.5 : Les interfaces essentielles dans android studio.

3.5. Découpage fonctionnel de l'application :

L'application est organisée autour de fragments réutilisables intégrés dans une activité principale (MainActivity). Voici les composants majeurs :

3.5.1. L'interface de Activity_login :

- Permet l'authentification sécurisée de l'utilisateur.
- Comprend une interface de connexion et une interface pour la création de compte.
- Garantit que seules les personnes autorisées accèdent aux données de la serre.
- Permet de créer un compte.

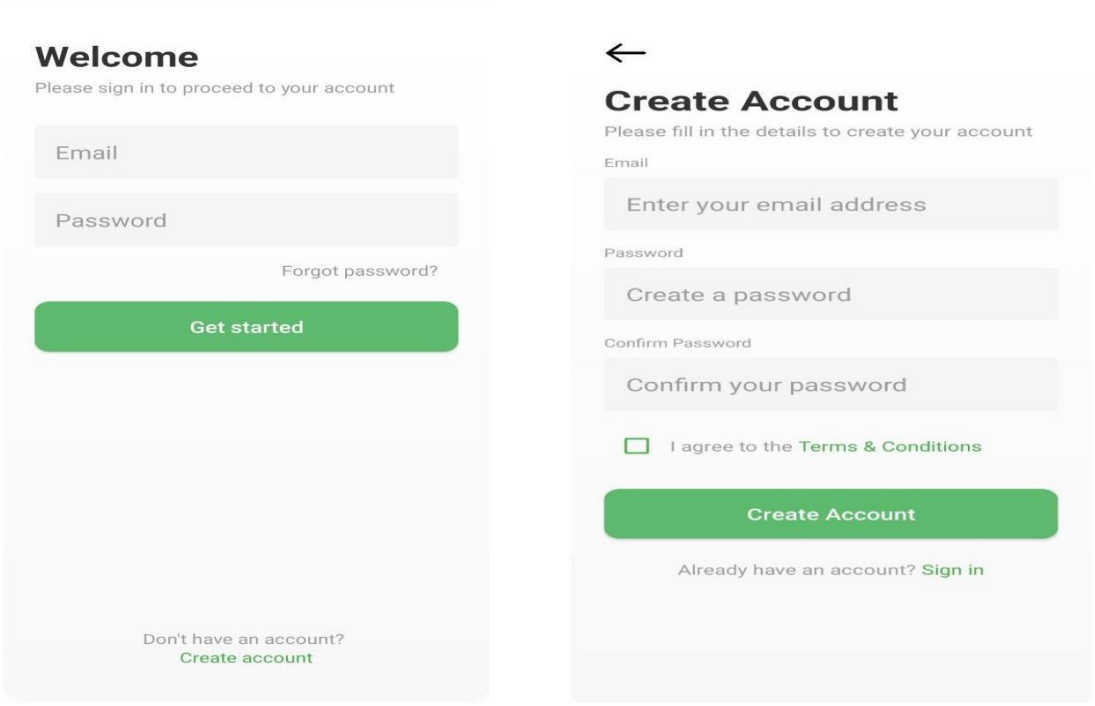


Figure 3.6 : les deux parties de l'interface login.

3.5.2. L'interface de Fragment main :

- Sert de container principal qui contient les autres fragments (via une barre de navigation).
- Gère la navigation fluide entre les écrans : Home, Room, Setup, Graphiques, Tools, Wi-Fi, AI.

Figure 3.7 : le fragment main.

3.5.3. L'interface de Fragment home :

C'est le premier fragment qui ne permet d'afficher les conditions météo.

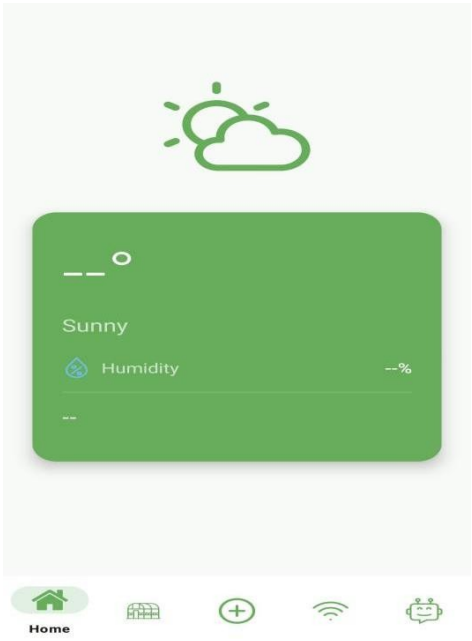


Figure 3.8 : l'interface home.

3.5.4. L'interface de Fragment Room :

Ce fragment permet à l'utilisateur d'organiser plusieurs chambres de produits, et affiche la date d'ajoute. Considérez toutes les appareils connectés, avec fonction de modification (setup) et afficher les listes qui nous permet de voir des interfaces pour les statistique pourcentage et graphique du chaque appareil, comme :

Room Name: Test1

Date added: May 12.2025

Device count : 0

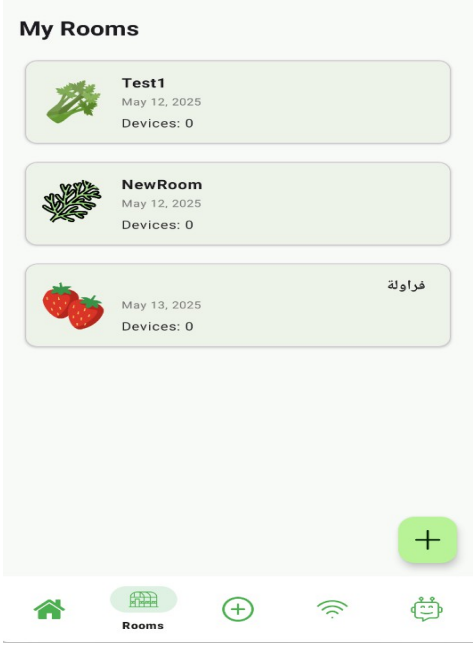


Figure 3.9 : l'interface de rooms.

3.5.5. Interface de fragment graphiques de chaque device :

Le fragment graphique représente visuellement l'interface de l'utilisateur ou le tableau de bord permettant de suivre en temps réel les données collectées par les capteurs. Il joue un rôle essentiel dans la visualisation intuitive des mesures environnementales température, humidité, luminosité, qualité de l'air.

Figure 3.10 : l'interface graphique.

3.5.6. L'interface de Fragment setup (add) :

Il permet d'ajouter des rooms et devices dans le système.

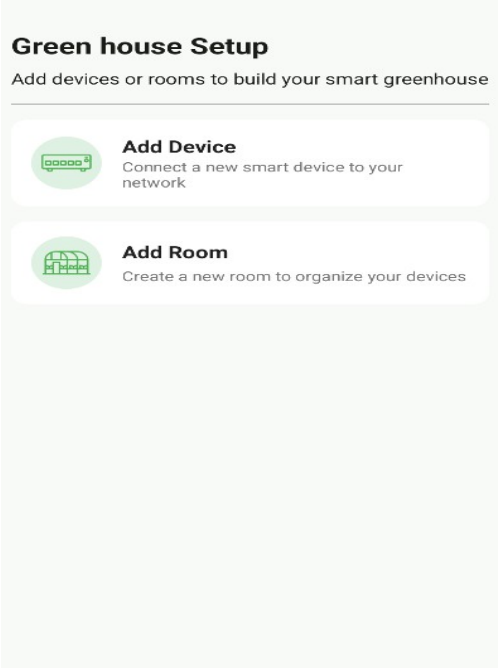


Figure 3.11 : l'interface setup.

La figure 3.12 montre l'interface du partie add device .

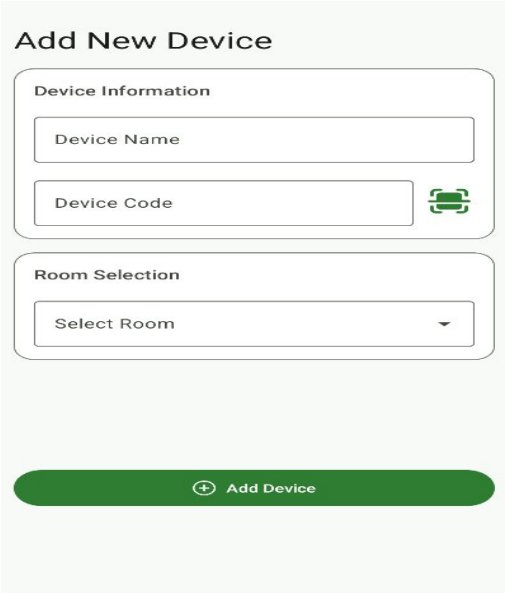


Figure 3.12 : l'interface de la partie add device.

Et la figure 3.13 montre l'interface de la partie add room.

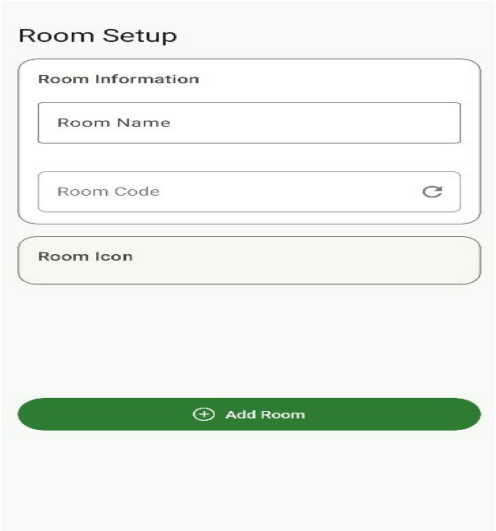


Figure 3.13 : l'interface du partie add room.

3.5.7. L'interface de Fragment l'API AI :

Ce fragment Permet d'interagir avec une intelligence artificielle intégrée (figure 3.14).

Figure 3.14 : l'interface de l'API AI.

3.5.8. L'interface de Fragment ESP32 wi-fi :

Ce fragment permet d'ajouter les réseaux wi-fi des ESP32 (figure 3.15).

Figure 3.15 : l'interface de wifi esp32.

3.5.9. L'interface de Fragment_tools :

Le Fragment_tools est une partie de l'application qui permet à l'utilisateur de contrôler les outils de la ferme intelligente. Depuis cet écran, l'utilisateur peut activer ou désactiver la led, la pompe à eau, le fan et le chauffage.

3.5.10. La structure de layout Fragment_tools :

La figure 3.16 represente la sturcture du fragment tools.

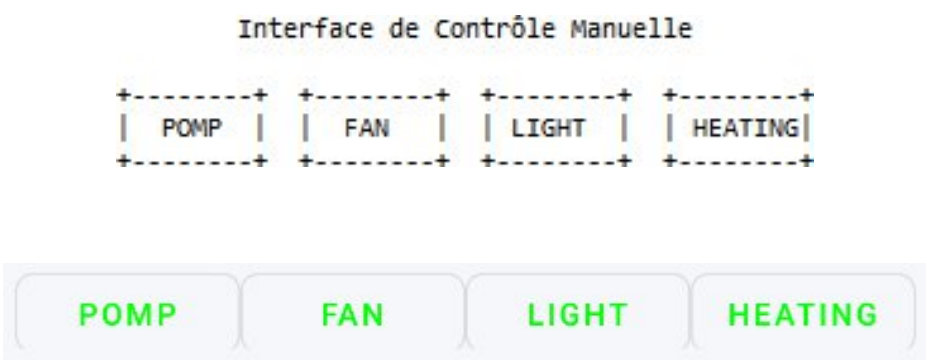


Figure 3.16 : l'interface de Fragment_tools.

3.6. Étapes pour lancer l'application :

Pour lancer l'application Android dans Android Studio il faut :

1-Connecte un téléphone ou démarre un émulateur.

2-Active le Mode développeur sur ton téléphone.

3-Active Débogage USB.

4-Crée un émulateur ou démarre-en un.

5-Choisis l'appareil sur lequel lance.

6-Clique sur le bouton Run pour le projet.

3.7. Les avantages de l'application :

L'application permet de surveiller ses cultures en temps réel et de contrôler facilement des choses comme l'arrosage ou la ventilation.....

Grâce aux capteurs, il peut économiser de l'eau et de l'énergie en utilisant seulement ce qui est nécessaire. L'application envoie aussi des alertes quand il y a un problème, ce qui permet d'agir rapidement. Le fermier peut consulter les données à tout moment, même s'il n'est pas sur place, ce qui lui fait gagner du temps et lui simplifie le travail.

3.8. Étude des technologies du site web :

3.8.1. HTML :

Le HyperText Markup Language (HTML) est le langage de balisage standard utilisé pour structurer les pages web. Dans notre projet, HTML a permis de définir les éléments principaux du site tels que, L'en-tête (header) contenant le logo et le menu de navigation, Les sections de contenu (à propos, solution, galerie, contact…) et Le pied de page (footer) .

3.8.2. CSS

Le Cascading Style Sheets (CSS) est utilisé pour styliser les éléments HTML et contrôler leur apparence sur différents supports. Le CSS a été utilisé pour définir les couleurs, polices, espacements, et tailles des éléments.

3.8.3. JavaScript

JavaScript est un langage de programmation orienté client permettant d'ajouter de l'interactivité aux pages web, Gérer les animations légères.

3.8.4. Node.js

Le backend du site a été développé avec Node.js, un environnement JavaScript côté serveur, combiné au framework Express.js. Il permet de gérer le traitement des formulaires, Communication avec une base de données, Création d'API REST.

3.9. Structure Générale d'un Site Web

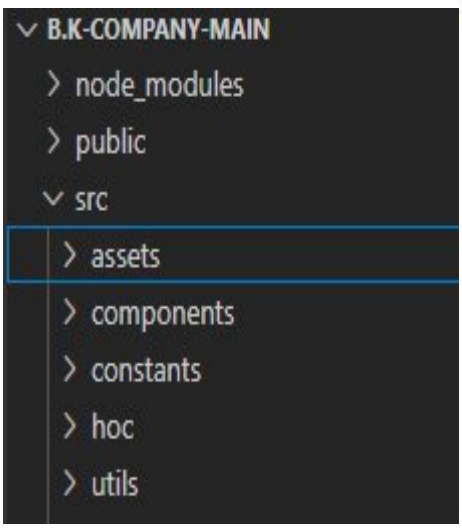
- Les interfaces utilisateur Partie visible pour l'utilisateur.
- Le Back-end partie serveur.

```
iot-agri-website/
├─ public/          # Fichiers statiques (HTML/CSS/JS)
│   ├─ index.html
│   ├─ style.css
│   └─ script.js
├─ server.js        # Backend (Node.js)
├─ package.json
└─ node_modules/
```

Figure 3.17 : Structure Générale d'un Site Web

3.9.1. Conception et architecture

Le site Web B.K est construit sur l'architecture React.js. React est une bibliothèque JavaScript développée par Facebook, qui permet de créer des interfaces utilisateurs dynamiques, module.




```
src/
├── components/
│   ├── About.jsx
│   ├── Contact.jsx
│   ├── Experience.jsx
│   ├── Feedbacks.jsx
│   ├── Hero.jsx
│   ├── Loader.jsx
│   ├── Navbar.jsx
│   ├── Tech.jsx
│   └── Works.jsx
├── constants/
├── hoc/
├── utils/
├── App.jsx
├── index.js
├── main.jsx
├── styles.js
└── index.css
```

Figure 3.18 : l'organisation des fichiers de création de site web.

3.9.1.1. Public

Contient les fichiers accessibles directement par le navigateur, comme index.html et les icônes du site. C'est la base statique du projet.

3.9.1.2. src

Dossier principal du code source de l'application React. Il est divisé en plusieurs sous-dossiers :

3.9.1.3. assets

Contient les ressources statiques du projet telles que les images, logos, icônes et fichiers multimédias utilisés dans le site.

3.9.1.4. Components

Contient tous les composants React correspondant aux différentes sections du site (Hero, About, Contact, etc.). Chaque composant est un module indépendant et réutilisable.

3.9.1.5. Constants

Contient les fichiers de données statiques (par exemple, listes de services, informations sur les technologies) utilisées par plusieurs composants pour éviter la duplication de code.

3.9.1.6. Hoc/ (High Order Components) :

Contient les composants d'ordre supérieur, c'est-à-dire des fonctions qui permettent d'enrichir ou de modifier d'autres composants, sans en changer la structure d'origine. Cela améliore la réutilisation et l'organisation du code.

3.9.1.7. utils :

Contient des fonctions utilitaires (helpers), comme des fonctions de formatage, de validation de données ou autres logiques récurrentes utilisées dans le projet.

3.9.1.8. node_modules :

Dossier généré automatiquement par NPM (Node Package Manager) contenant toutes les dépendances nécessaires au projet.

3.10. Node Package Manager (npm)

Sur le bash **:** npm instal, Est un outil indispensable utilisé dans les projets Node.js et React.js pour gérer les dépendances,il permet de télécharger et installer automatiquement les bibliothèques ou modules nécessaires au projet (comme React, Express, TailwindCSS) et Gérer la version de chaque module pour assurer la compatibilité du projet.

3.10.1. Style CSS :

Dans notre projet il y a un fichier principal index.css centralise les styles globaux appliqués à tout le site.

Certains composants utilisent également des styles en JavaScript fichier **styles.js**, une méthode appelée CSS-in-JS, pratique pour styliser directement un composant React.

3.10.2. Les composants du React

Est une partie indépendante et réutilisable de l'interface utilisateur. Chaque composant représente une section précise du site.

3.10.2.1. Section Hero.jsx

La section Hero.jsx (figure 3.21) est un composant React qui représente la partie d'accueil principale d'un site web. C'est souvent la première section que l'utilisateur voit lorsqu'il arrive sur le site. Elle contient des informations sur nous pour attirer l'attention

- **Contenu typique :**

- Un titre accrocheur.
- Une phrase de description courte.

- Des boutons d'action.
- Une image et illustration de la serre connectée avec du système.

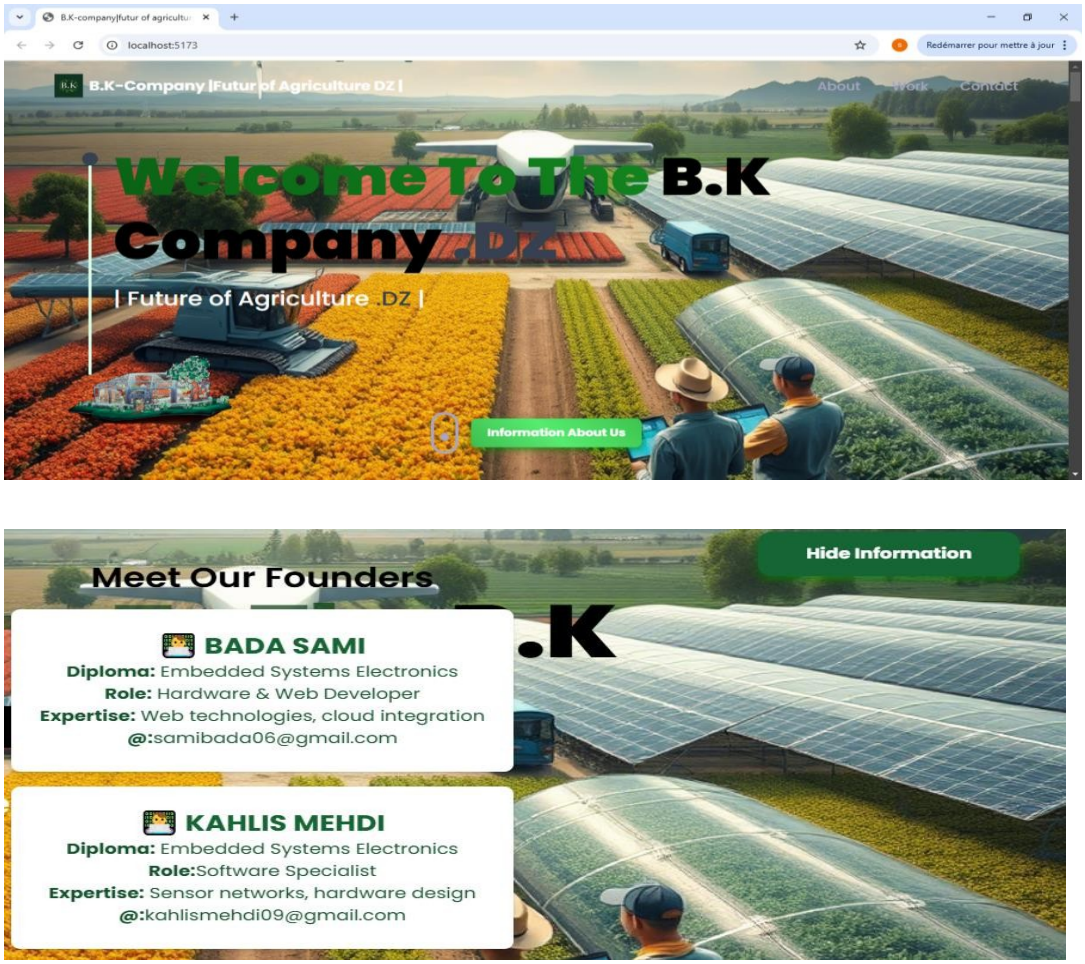


Figure 3.19: la section Hero du site web.

La figure 3.20 représente La structure de section Hero

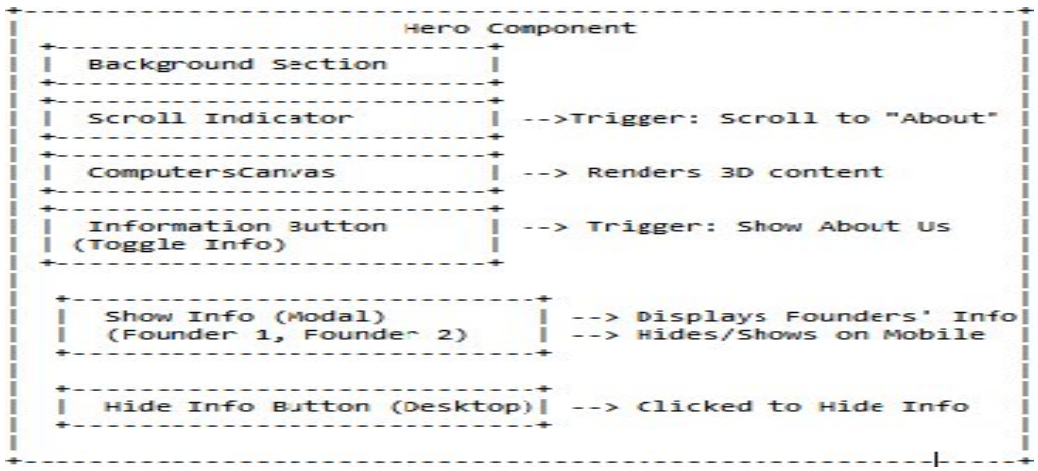


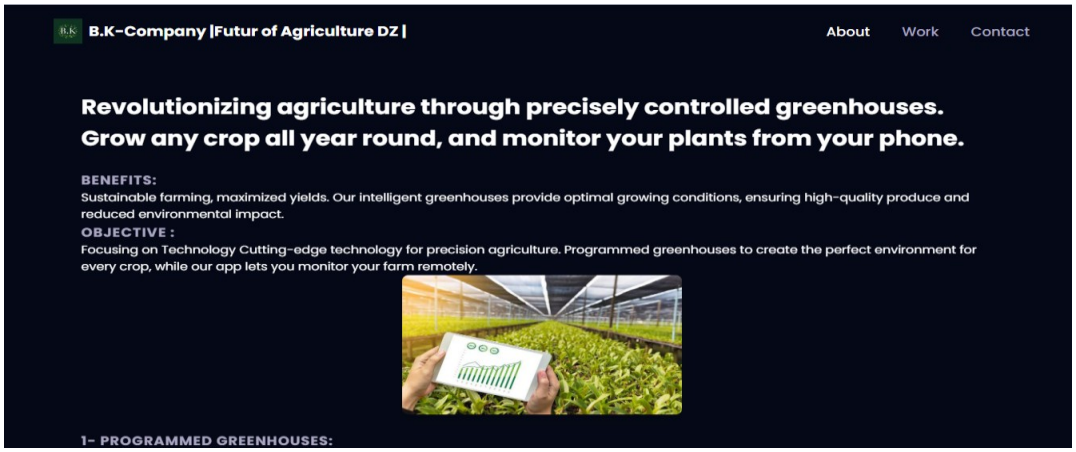
Figure 3.20 : La structure de section Hero .

3.10.2.2. Section About.jsx

La section About représente la partie de l'explication sur notre mission et notre projet.

➢ **Contenu typique**

- Une courte présentation de B.K COMPANY.
- Pourquoi nous avons créé ce système.
- Des photos du matériel utilisé.



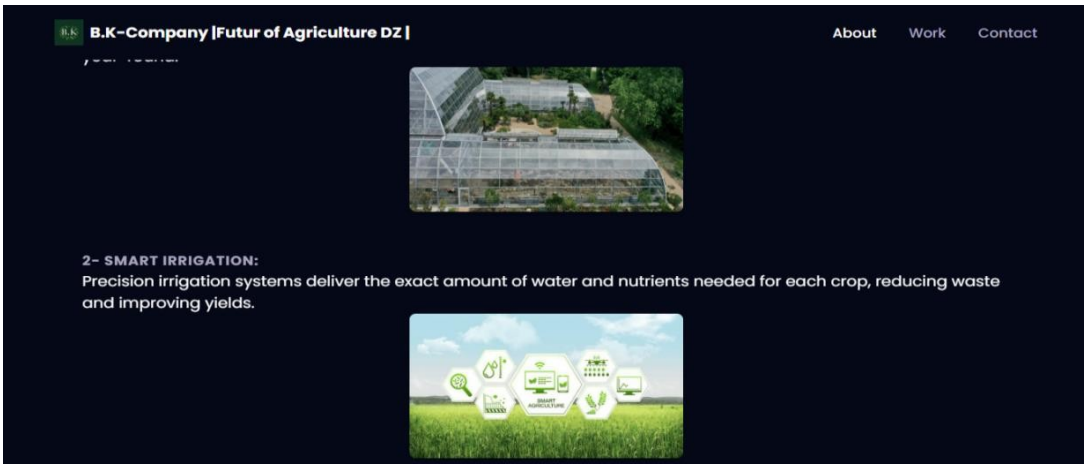


Figure 3.21 :la section about du site web

La figure 3.22 montre la structure de section About.

```
About (wrapped in SectionWrapper)
│
├── motion.div (Intro Texte)
│   ├── Titre principal (Revolutionizing agriculture...)
│   ├── Sous-titres (Benefits, Objectives)
│
├── motion.div (Contenu principal)
│   ├── Image: evolution.png
│   ├── Section 1: Programmed Greenhouses
│   │   └── Image: house.png
│   ├── Section 2: Smart Irrigation
│   │   └── Image: smart-integration.png
│   ├── Section 3: LED Lighting
│   │   └── Image: LED-lights.png
│   ├── Section 4: AI-Powered Monitoring
│   │   └── Image: IA.png
│   └── Section: Additional Services for Investors
│
└── Service Cards (map sur services[])
    ├── ServiceCard 1 (icon + title)
    ├── ServiceCard 2
    ├── ...
    └── ServiceCard n
        └── Utilise: Tilt + motion.div (animé avec fadeIn)
```

Figure 3.22 : la structure de section About.

3.10.2.3. Section Works.jsx

La section Works montrer ce que fait votre système IoT, avec des exemples concrets.

- **Contenu typique :**
- Le but de notre projet.
- Des explications sur les projets qui don dans les recherches.

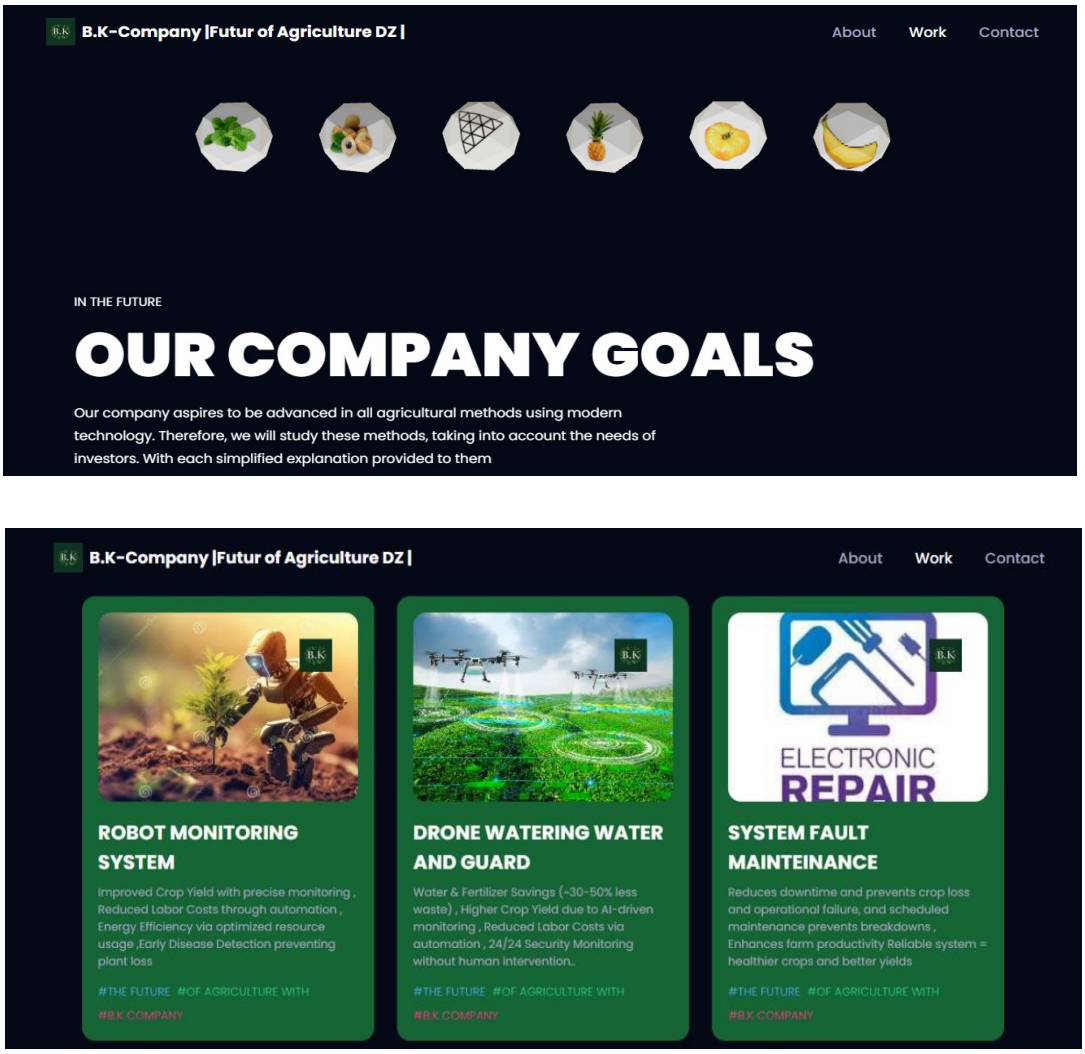


Figure 3.23 :La section Works du site web.

La structure de section Work est représenté dans la figure 3.24.

```
Works |
│
├── motion.div (Header)
│    ├── Subtitle: "IN THE FUTURE"
│    └── Title: "OUR COMPANY GOALS"
│
├── motion.p (Intro Texte)
│    └── Explication de la vision de l'entreprise
│
└── Projects List (map sur projects[])
     ├── ProjectCard 1 (props: name, desc, image, tags, link)
     │    ├── motion.div (fade-in up)
     │    └── Tilt container (3D effect)
     │         ├── Image (project thumbnail)
     │         ├── GitHub Button (ouvre le code source)
     │         ├── Title & Description
     │         └── Tags (affichés avec `#tag`)
     ├── ProjectCard 2
     └── ...
```

Figure 3.24 : La structure de section Work.

3.10.2.4. Section Experience.jsx

La section Experience fournir des informations sur les plantes et faire des présentations aux investisseurs

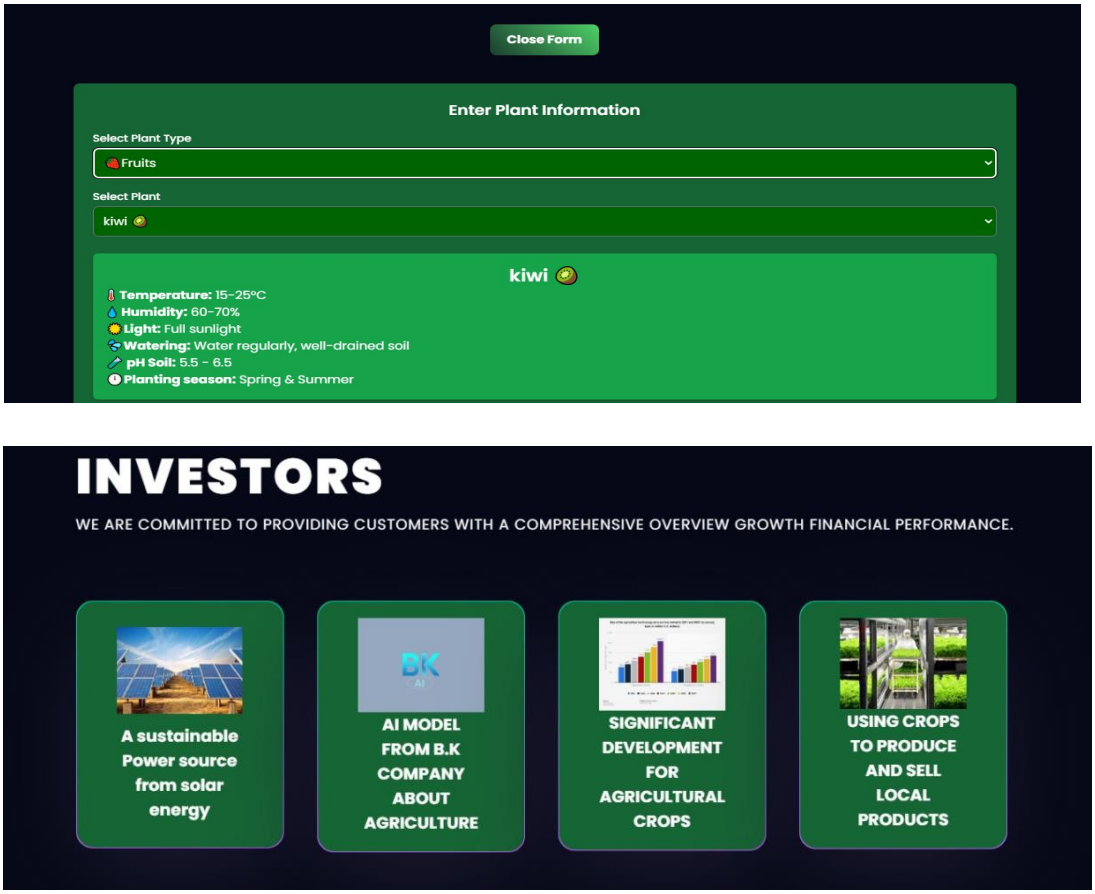


Figure 3.25 : la section Expérience du site web.

Alors que la structure de section Expérience est représenté dans la figure 3.26.

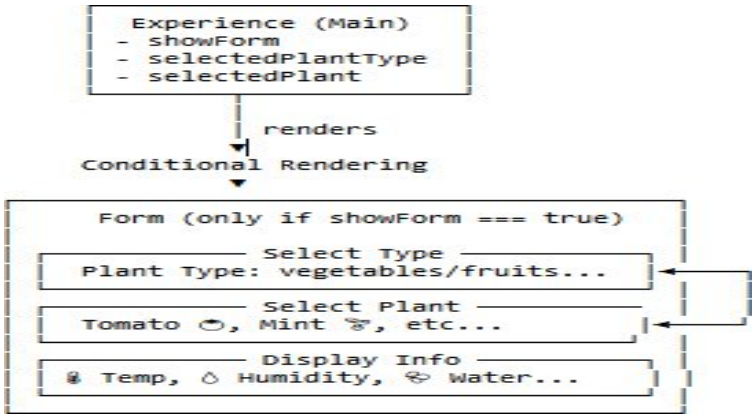


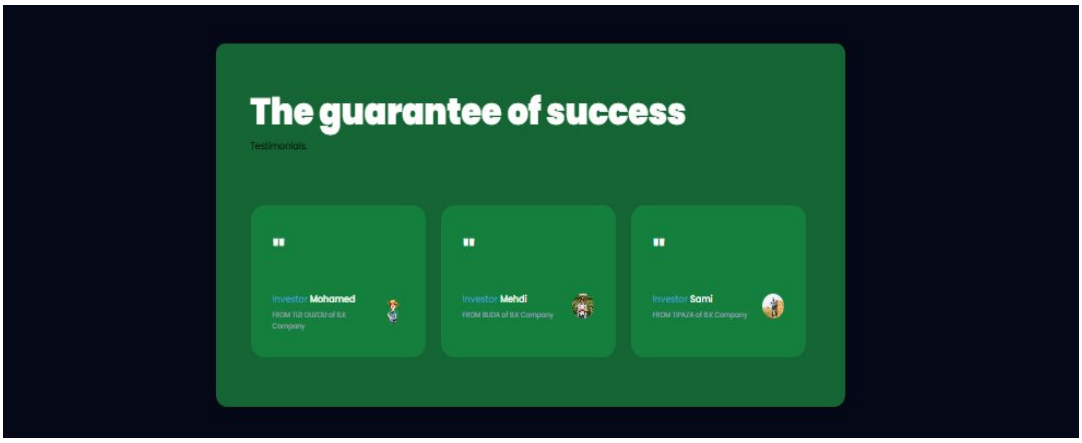
Figure 3.26 : la structure de section Expérience

3.10.2.5. Section contact.jsx

Permettre aux visiteurs de vous joindre facilement, et demander des infos.

- **Contenu typique :**

- Un formulaire de contact (nom, e-mail, message).
- Liens vers les réseaux sociaux(help).
- Un formulaire de garantie.



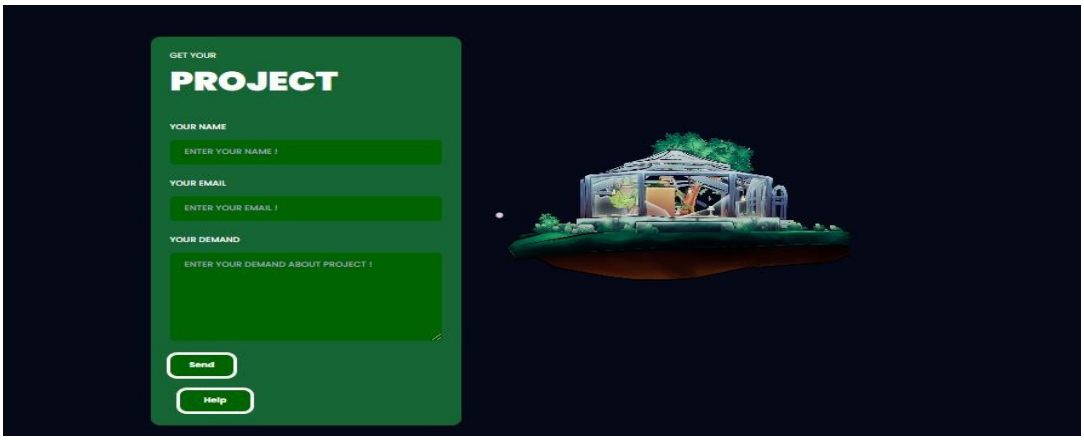


Figure 3.27 : la section contact du site web.

La figure 3.28 représente la structure de section Contact.

```
Contact (composant React)
│
├── useRef() → formRef  (référence au formulaire HTML)
├── useState()
│   ├── form: { name, email, message } (valeurs des champs du formulaire)
│   └── loading: boolean (état d'envoi du message)
│
├── Fonctions internes
│   ├── handleChange(e)
│   │   └── met à jour `form` selon input.name et input.value
│   └── handleSubmit(e)
│       ├── empêche rechargement de la page
│       ├── set loading = true
│       ├── utilise emailjs.send() pour envoyer les données
│       ├── en cas de succès : reset form et affiche un message
│       └── en cas d'erreur : affiche une alerte d'erreur
│
├── Formulaire HTML
│   ├── Champ input: name
│   ├── Champ input: email
│   ├── Champ textarea: message
│   ├── Bouton [Send]
│   └── Bouton [Help] → redirige vers epicgardening.com
│
├── Styles et animation
│   ├── motion.div → animations de la section (slideIn)
│   └── EarthCanvas → composant de visualisation 3D
│
└── Exportation : export default SectionWrapper(Contact, "contact")
    └── HOC qui entoure le composant avec une mise en page ou un style global
```

Figure 3.28 : la structure de section Contact

3.11. l'Email.js

L'EmailJS est un service en ligne qui te permet d'envoyer des e-mails directement depuis le site web, sans avoir besoin de backend ou de serveur SMTP.

a. Créer un compte sur emailjs.com.

b. Ajouter un service email.

c. Créer un template d'email.

- Un ID de service.
- Un ID de template.
- Une clé publique API.

EmailJS Integration

```
emailjs.send(
  "service_XXXXX",
  "template_XXXXX",
  {
    from_name: "Ton nom",
    message: "Ton message",
    // autres variables du template...
  },
  "clé_publique"
)
```

Figure 3.29 : la structure de code pour configurer l'Email.js.

Comme le diagramme dans la figure 3.30.

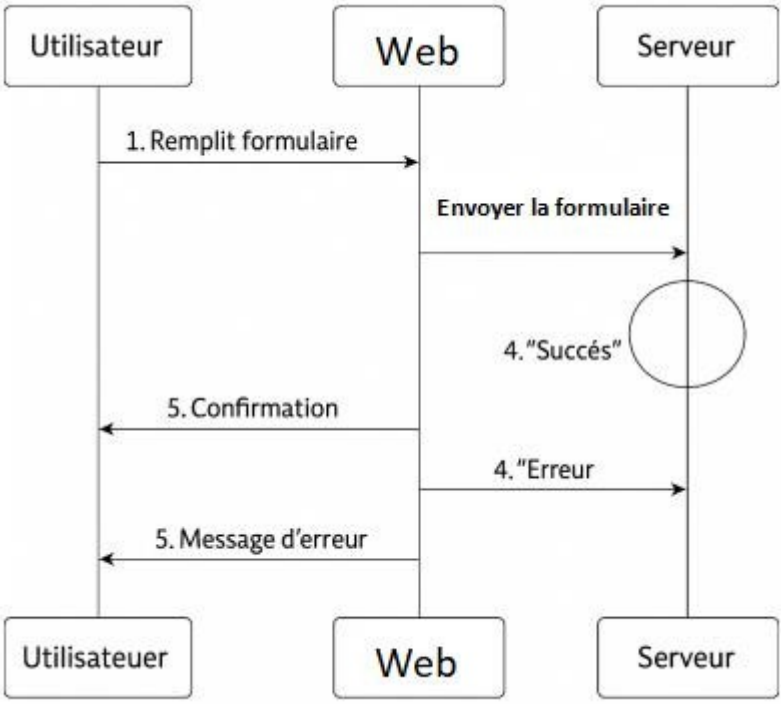


Figure 3.30 : Diagramme de fonctionnement email.js.

3.12. Conclusion

Les interfaces de l'application mobile et du site web vitrine jouent un rôle essentiel dans la réussite d'un système IoT agricole. Les interfaces de l'application permettent aux utilisateurs d'interagir facilement avec la technologie, de visualiser les données environnementales en temps réel, et de contrôler les équipements à distance. . Le site web vitrine, quant à lui, valorise le projet, présente ses avantages et renforce la crédibilité de l'entreprise. Ensemble, ces interfaces sont la passerelle entre l'utilisateur et la technologie, et participent directement à la diffusion, la compréhension et l'impact de la solution IoT.

4. Chapitre 4 : Interaction du Fragment avec le serveur et résultats

4.1. Introduction :

Dans le chapitre précédent, nous avons présenté l'architecture et le rôle du serveur dans notre application, en tant qu'intermédiaire entre les dispositifs ESP32 et l'application mobile. Ce serveur gère non seulement les fonctionnalités d'authentification (connexion et création de compte), mais fournit également des informations en temps réel et historiques (journalières et hebdomadaires) sur les capteurs connectés à chaque dispositif.

Dans ce chapitre, nous allons détailler le mécanisme d'interaction entre les composants de l'interface utilisateur (Fragments) et le serveur. Cette interaction est rendue possible grâce à une classe centrale nommée **'**BaseHandler', qui assure la gestion de la communication TCP via un thread dédié (ExecutorService). Cette classe se charge de recevoir les paquets envoyés par le serveur, de les interpréter en fonction de leur opcode, et de déclencher les fonctions appropriées à travers un système de gestion des handlers.

Pour transmettre les données reçues aux Fragments concernés, nous utilisons le mécanisme de diffusion locale ('LocalBroadcastManager'). Ce choix permet de découpler la logique réseau de la couche interface, tout en assurant une communication fluide et réactive entre les différentes composantes de l'application. Les Fragments peuvent ainsi recevoir des mises à jour sur l'état des capteurs, le résultat des opérations de connexion, ou l'ajout d'un nouvel appareil, et ce, sans dépendre directement de la couche réseau.

Ce chapitre présente donc le cycle complet de l'information, depuis sa réception par le serveur, jusqu'à son affichage ou traitement par les Fragments, en mettant en lumière les choix techniques adoptés pour garantir la modularité et la maintenabilité de l'application.

4.2. Classe de communication (BaseHandler) avec le serveur via TCP :

Cette classe joue un rôle central dans l'application en assurant la liaison continue et fiable entre le client et le serveur via le protocole TCP. Elle est responsable de l'établissement de la connexion, de

l'envoi et de la réception des données, ainsi que de la gestion des différents types de messages selon leurs codes d'opération (opcodes).

Grâce à une architecture basée sur l'utilisation de sockets et de threads, cette classe garantit que les données reçues du serveur sont traitées de manière asynchrone et distribuées aux différentes fonctions spécifiques en fonction du contenu reçu.

Dans la suite, nous détaillerons les étapes clés de son fonctionnement, depuis la connexion initiale jusqu'à la réception et le traitement des données.

4.2.1. Établissement de la connexion au serveur (Connection Setup) :

Le processus de connexion au serveur dans la classe commence par une série d'étapes claires pour garantir une connexion stable et fiable. Ces étapes comprennent l'initialisation du socket, la configuration de l'adresse du serveur, l'ouverture de la connexion, et la gestion des erreurs éventuelles.

4.2.1.1. Initialisation du socket :

- **Définition du socket :** Le socket est le point d'extrémité permettant la communication réseau. Il sert d'interface pour l'envoi et la réception de données via le protocole TCP/IP.
- **Création du socket :** La classe crée un nouveau socket en spécifiant la famille d'adresses IPv4 et le type TCP, assurant ainsi une communication fiable et orientée connexion.
- **Configuration des options du socket :** Certaines options peuvent être définies sur le socket, telles que le mode non-bloquant (non-blocking) ou le délai d'attente (timeout), afin de contrôler le comportement de la connexion.

4.2.1.2. Configuration de l'adresse du serveur (IP + Port) :

L'adresse du serveur est spécifiée par :

- **Adresse IP :** l'adresse IP du serveur (par exemple : 194.164.77.65).
- **Numéro de port :** le port sur lequel le serveur écoute (par exemple : 57005).

```
public static final String SERVER_IP = "194.164.77.65";
public static final int Port = 57005;
```

Figure 4.1 : la définition de l'adresse IP et le Numéro de port du serveur dans la classe .

Ces informations sont soit définies directement dans la classe, soit passées en paramètres lors de la création de la connexion.

4.2.1.3. Ouverture de la connexion (connect) :

- En utilisant le socket et l'adresse configurée, la méthode connect() est appelée pour établir une session TCP entre le client et le serveur.
- En cas de succès, la connexion est établie et la classe peut commencer à recevoir des données.

4.2.1.4. Gestion des erreurs de connexion (Retry, Timeout) :

- **Timeout :** Si la tentative de connexion dépasse la durée maximale définie, elle est considérée comme échouée.
- **Retry (reconnexion) :** En cas d'échec (serveur indisponible, coupure réseau, etc.), la classe peut retenter la connexion automatiquement après un délai défini (par exemple, 5 secondes).
- **Gestion des exceptions :** Les erreurs système ou logiques rencontrées durant la connexion doivent être capturées afin d'informer les autres parties du programme et permettre une réaction appropriée (notification utilisateur, nouvelle tentative, etc.).

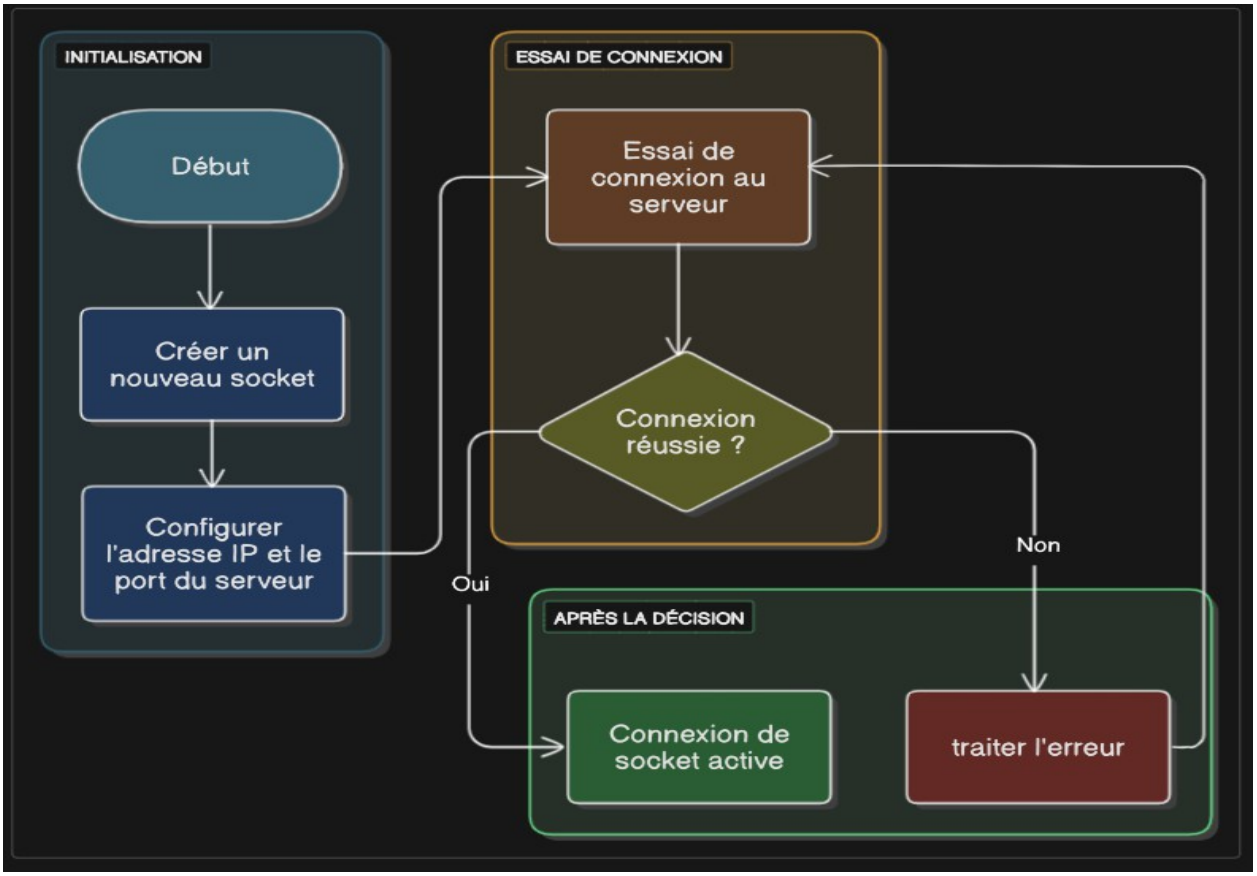


Figure 4.2 : Schéma de l'établissement de la connexion TCP.

4.2.2. Réception des données depuis le serveur :

Une fois la connexion TCP établie, la classe de communication démarre un thread dédié à la réception continue des paquets envoyés par le serveur. Cette réception se fait de manière structurée en suivant **les étapes suivantes :**

4.2.2.1. Lecture de la taille du paquet

La méthode commence par lire exactement 2 octets depuis le flux d'entrée (InputStream.read). Ces deux octets représentent la taille totale du paquet à recevoir, encodée sous forme de nombre short avec l'ordre d'octets Little Endian.

4.2.2.2. Lecture du contenu du paquet

Une fois la taille extraite, la méthode lit exactement ce nombre d'octets depuis le flux. Cela permet d'obtenir l'intégralité du paquet.

4.2.2.3. Décodage de l'opcode

Les 2 premiers octets du paquet reçu correspondent à un identifiant appelé Opcode, qui indique le type du message.

4.2.2.4. Distribution logique du paquet

Le Opcode est ensuite utilisé dans une structure conditionnelle (switch-case) pour appeler la méthode spécifique qui saura interpréter les données restantes du paquet. Chaque Opcode est ainsi associé à une fonction de traitement dédiée.

4.2.2.5. Gestion des erreurs et de la fermeture du flux

Si une erreur survient pendant la lecture (par exemple : fermeture du flux, exception réseau), le thread de réception est stoppé et une trace d'erreur est affichée pour faciliter le diagnostic.

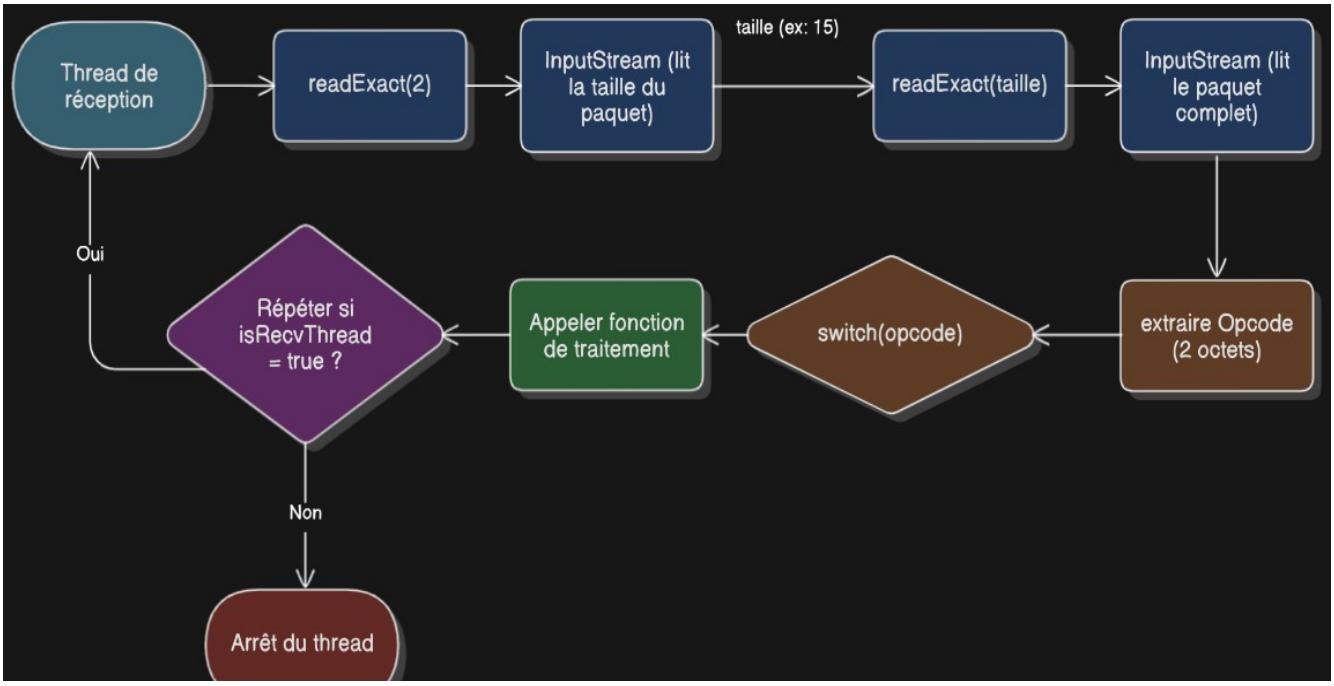


Figure 4.3 : Diagramme de réception d'un paquet TCP

À noter :

- La boucle de réception continue tant que le flag isRecvThread est activé.
- La méthode readExact(int count) garantit la lecture exacte du nombre d'octets requis, en effectuant plusieurs lectures partielles si nécessaire.

4.3. Transmission et réception des données via LocalBroadcast dans le Fragment

Dans cette section, nous allons détailler le mécanisme de communication entre la classe réseau principale (responsable de la connexion et de la réception des paquets TCP) et les différentes interfaces utilisateur représentées par les Fragments.

Pour garantir une séparation claire entre la logique réseau et l'interface graphique (UI), l'application utilise le système LocalBroadcastManager d'Android. Celui-ci permet d'envoyer des messages internes entre composants de l'application sans créer de dépendances directes entre eux.

Chaque Fragment :

- Envoie des requêtes au serveur via des fonctions spécifiques.

- Attend des réponses sous forme de paquets.

- Reçoit les données via un récepteur ('BroadcastReceiver') enregistré localement.

- Met à jour l'interface utilisateur en conséquence.

Nous allons maintenant présenter chaque Fragment, ses responsabilités, les données qu'il envoie au serveur et celles qu'il reçoit.

4.3.1. Comportement commun des Fragments

Tous les Fragments de l'application partagent une structure de base similaire pour garantir une intégration fluide avec la logique réseau et une interface utilisateur réactive. Ce comportement commun se résume comme suit :

4.3.1.1. Initialisation des composants de l'interface (initViews)

Chaque Fragment commence par initialiser ses vues via une méthode initViews() dédiée. Cela permet d'associer les éléments de l'interface (boutons, textes, graphiques, etc.) aux variables correspondantes.

4.3.1.2. Définition des actions utilisateur (setOnClickListener)

Les événements utilisateurs tels que les clics sur les boutons sont gérés via des setOnClickListener qui déclenchent des fonctions spécifiques. Ces actions peuvent envoyer des requêtes au serveur ou effectuer des traitements locaux.

4.3.1.3. Enregistrement d'un BroadcastReceiver local

Chaque Fragment enregistre un BroadcastReceiver avec une action unique (ex : " SensorInfo", " Room_List_Fragment", etc.)

Afin d'écouter les réponses du serveur transmises via LocalBroadcastManager.

Fragment	BroadcastReceiver
AddDeviceFragment	Room_Add_Device
CreateAccountActivity	CreateAccountResponse
LoginActivity	Login_Status
RoomListFragment	Room_List_Fragment
RoomSetupFragment	Room_Setup_Fragment
SensorDashboardFragment	SensorInfo

Tableau 4.1 : Table de correspondance entre les Fragments et leurs BroadcastReceiver.

Cela permet une séparation claire entre la logique de communication réseau et l'affichage des données.

4.3.1.4. Désenregistrement lors de la destruction

Pour éviter les fuites de mémoire ou les comportements inattendus, chaque Fragment s'assure de désenregistrer son BroadcastReceiver lors de sa destruction (onDestroyView ou onDestroy).

Cette structure modulaire facilite la maintenance du code, la réutilisabilité, ainsi que la compréhension globale de la logique de transmission des données dans l'application.

4.3.2. Réception et affichage des pièces dans RoomListFragment

4.3.2.1. Fonctionnement général

Lors de l'ouverture du fragment RoomListFragment, une requête est envoyée au serveur pour obtenir la liste des pièces et des appareils associés. Cette opération se déroule selon les étapes suivantes :

a. Envoi au serveur du code d'opération **55** pour demander les données.
b. Réception d'une réponse avec le code **57**.
c. Traitement des données dans la fonction **process_room_info** de la classe **BaseHandler**.
d. Diffusion locale (**LocalBroadcast**) vers le fragment.
e. Affichage des pièces et appareils dans l'interface utilisateur.

4.3.2.2. Détail du traitement dans BaseHandler

À la réception de la réponse 57, la fonction process_room_info traite les données reçues sous forme binaire. Voici le format de la trame :

1. **Structure des données :**

- 2 octets (short) : nombre total de pièces.
- Pour chaque room:
    - 30 octets : nom de la pièce (string).
    - 30 octets : code de la pièce (string).
    - 1 octet: nombre d'appareils.
    - 4 octets : date de création (timestamp).
    - 4 octets: ID de l'icône.

- Pour chaque appareil :
  - 30 octets: nom de l'appareil.
  - 30 octets: code de l'appareil.
  - 1 octet : état de l'appareil (connecté ou non).

Chaque room est ajoutée à une liste locale roomList, avec ses appareils.

4.3.2.3. Diffusion vers le fragment :

Une fois les données traitées, un Intent est envoyé localement :

```
Intent intent = new Intent( action: "Room_List_Fragment");
intent.putExtra( name: "Reload",  value: "Reload");
LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Figure 4.4 : Intent pour afficher les données.

Cela permet à RoomListFragment de recevoir la mise à jour et d'afficher dynamiquement les données à l'utilisateur.

4.3.2.4. Fonctionnement dans le fragment :

Dans RoomListFragment :

- Le fragment enregistre un BroadcastReceiver avec le filtre "Room_List_Fragment".
- Lors de la réception, il déclenche une fonction pour rafraîchir l'interface utilisateur avec la nouvelle liste de pièces.

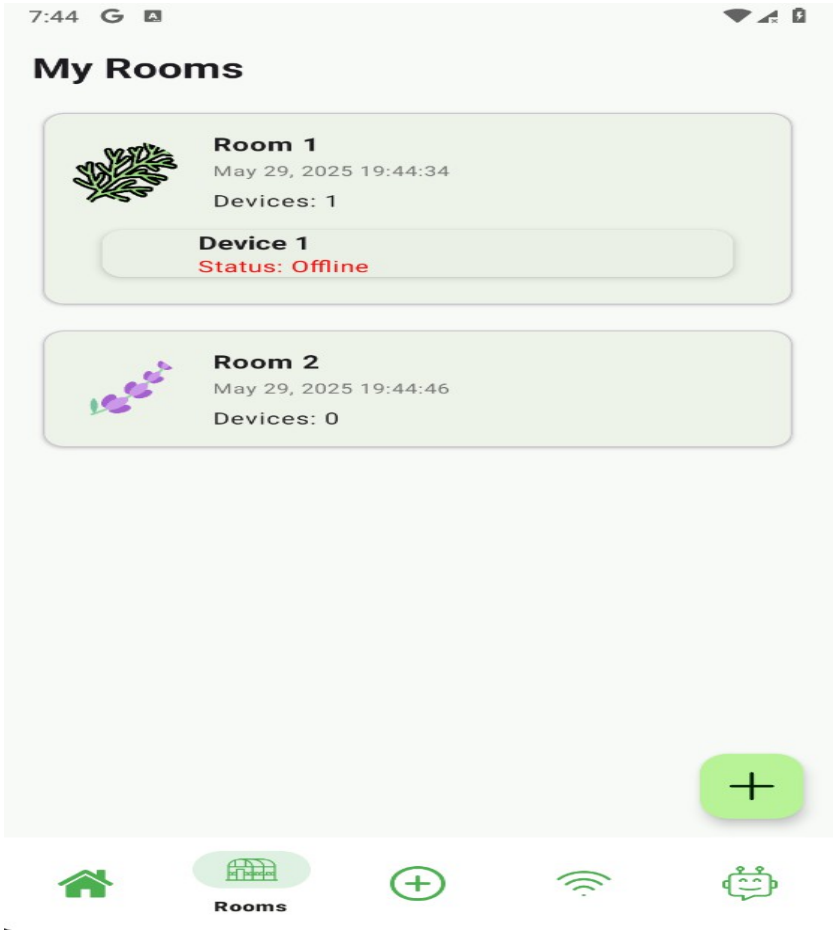


Figure 4.5 : Room list fragment résultat.

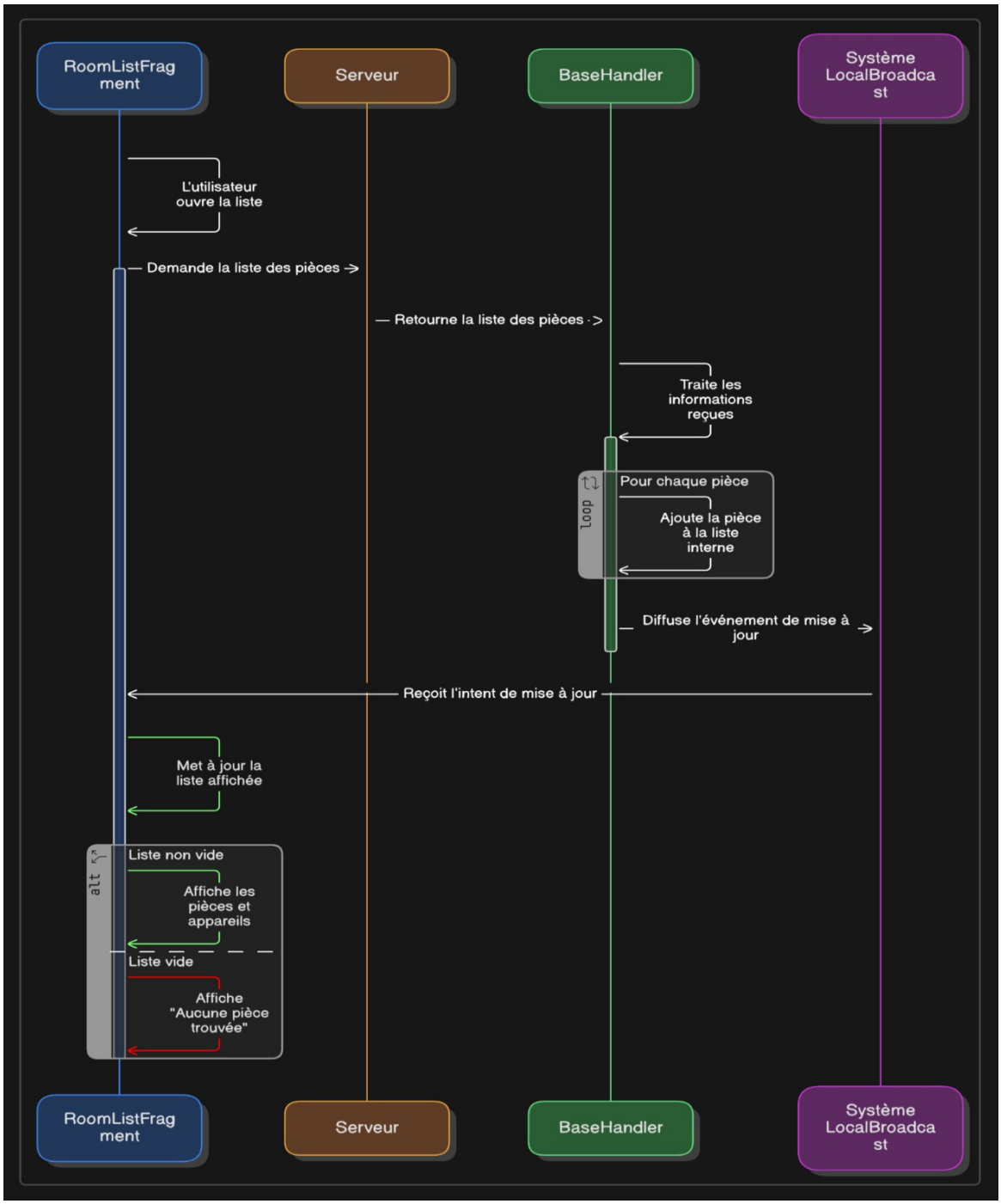


Figure 4.6 : Schéma de réception des pièces dans RoomListFragment.

4.3.3. Traitement de la connexion dans LoginActivity :

4.3.3.1. Déclenchement de la demande de connexion :

Lors de l'appui sur le bouton de connexion, une requête est envoyée au serveur avec l'opcode 1. La structure de la requête est la suivante :

- 30 octets: nom d'utilisateur
- 30 octets: mot de passe

4.3.3.2. Réception de la réponse du serveur :

Le serveur répond avec un paquet ayant l'opcode 2. Ce paquet est traité par la méthode process_login_status dans la classe BaseHandler, et contient :

- 1 octet : statut de la connexion (0 = échec, 1 = succès)

4.3.3.3. Transmission via LocalBroadcast :

Une fois le statut extrait, il est transmis à l'activité LoginActivity à l'aide d'un Intent :

```
Intent intent = new Intent( action: "LoginActivity");
intent.putExtra( name: "Login_Status", (int) status);
LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Figure 4.7 : Déplacer le statut vers LoginActivity.

4.3.3.4. Traitement dans LoginActivity :

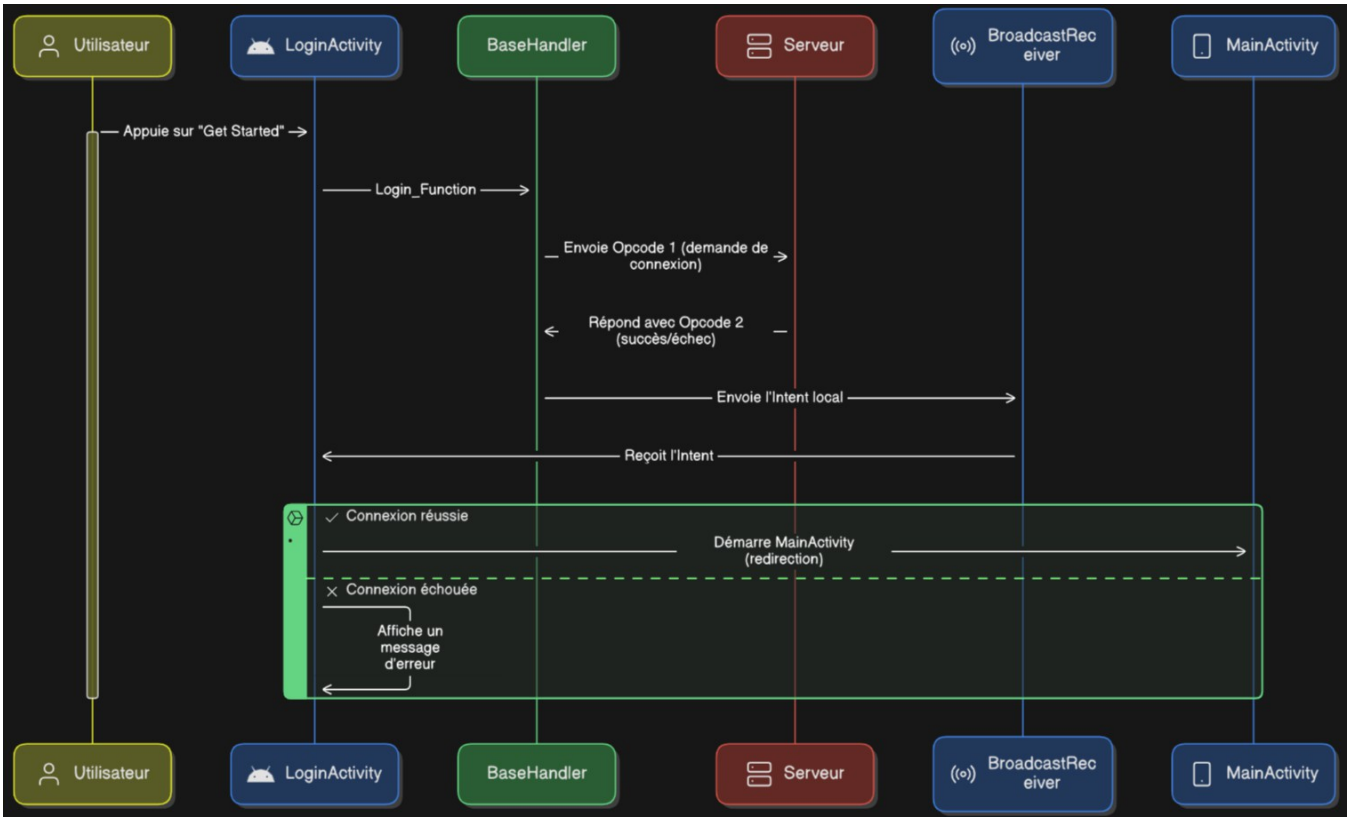


Figure 4.8 : Schéma de la gestion de la connexion dans LoginActivity

Le fragment écoute ce broadcast avec un BroadcastReceiver. Selon le statut reçu:

- En cas de succès (1) **:** l'utilisateur est redirigé vers MainActivity.
- En cas d'échec (0) : un message d'erreur est affiché (figure 4.9).

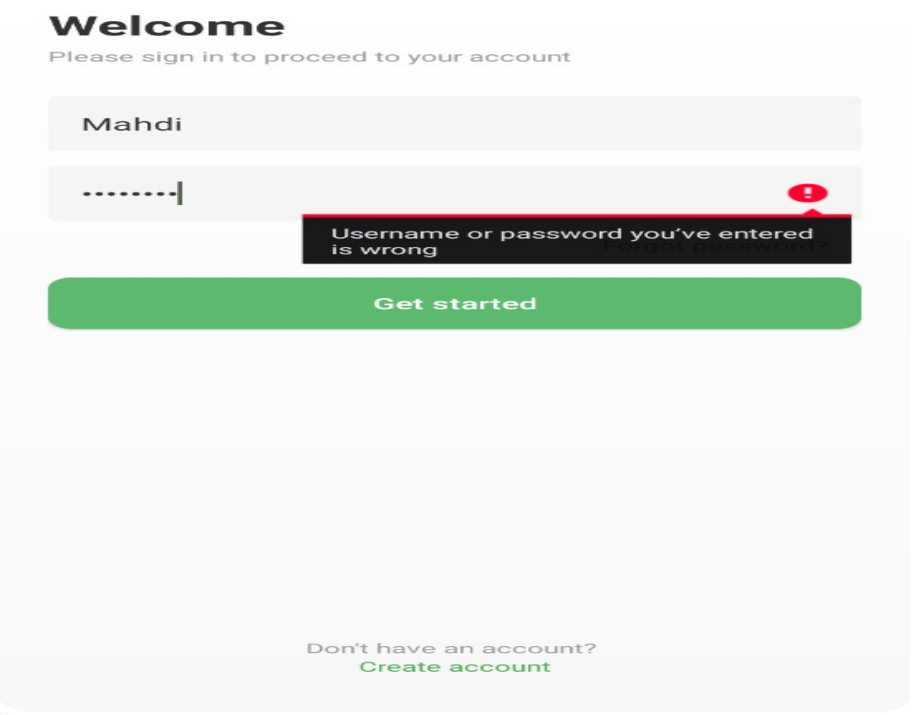


Figure 4.9 : LoginActivity resultat (0).

4.3.4. Gestion de la création de compte dans CreateAccountActivity :

Le fragment CreateAccountActivity permet à l'utilisateur de créer un nouveau compte. Il envoie les informations saisies au serveur via TCP, puis traite la réponse reçue pour afficher un message approprié.

4.3.4.1. Envoi de la requête :

Lors d'un clic sur le bouton Create Account, une requête est envoyée au serveur.

- **Opcode** utilisé : 13

2. **Contenu envoyé** :
   - 30 octets pour le nom d'utilisateur.
   - 30 octets pour le mot de passe.

4.3.4.2. **Réponse du serveur** :

Le serveur retourne une valeur entière (1 octet) correspondant à l'état du processus :

- **1 :** Le nom d'utilisateur ou le mot de passe est trop court (< 5 caractères).
- **2 :** Le nom d'utilisateur est déjà utilisé.
- **3 :** Le compte a été créé avec succès.

4.3.4.3. **Traitement dans BaseHandler** :

La fonction process_create_account_response lit la réponse et effectue une diffusion locale :

```
Intent intent = new Intent( action: "RegisterActivity");
intent.putExtra( name: "CreateAccountResponse", (int) response);
LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Figure 4.10 : Déplacer la réponse vers CreateAccountActivity.

4.3.4.4. **Réception dans CreateAccountActivity :**

Le fragment enregistre un BroadcastReceiver pour l'action "RegisterActivity".

Il extrait le code de réponse :

```
int status = intent.getIntExtra( name: "CreateAccountResponse",  defaultValue: -1);
```

Figure 4.11 : Déplacer la réponse vers CreateAccountActivity.

Il affiche ensuite un message utilisateur avec Toast :

- **1 :** Nom ou mot de passe trop court.
- **2 :** Nom d'utilisateur déjà utilisé.
- **3 :** Compte créé avec succès.

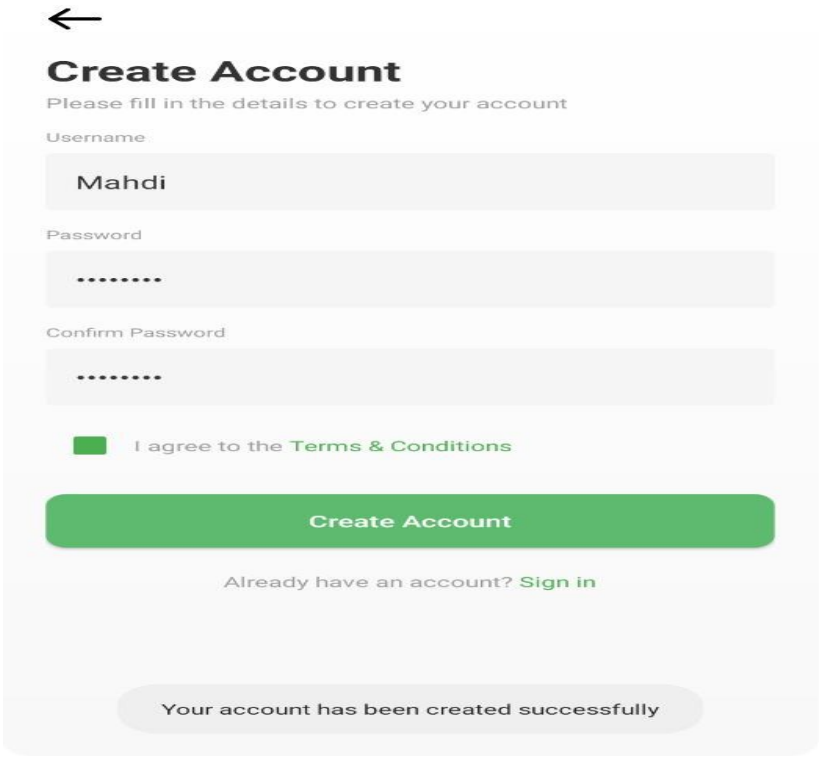


Figure 4.12 : CreateAccountActivity résultat (état 3).

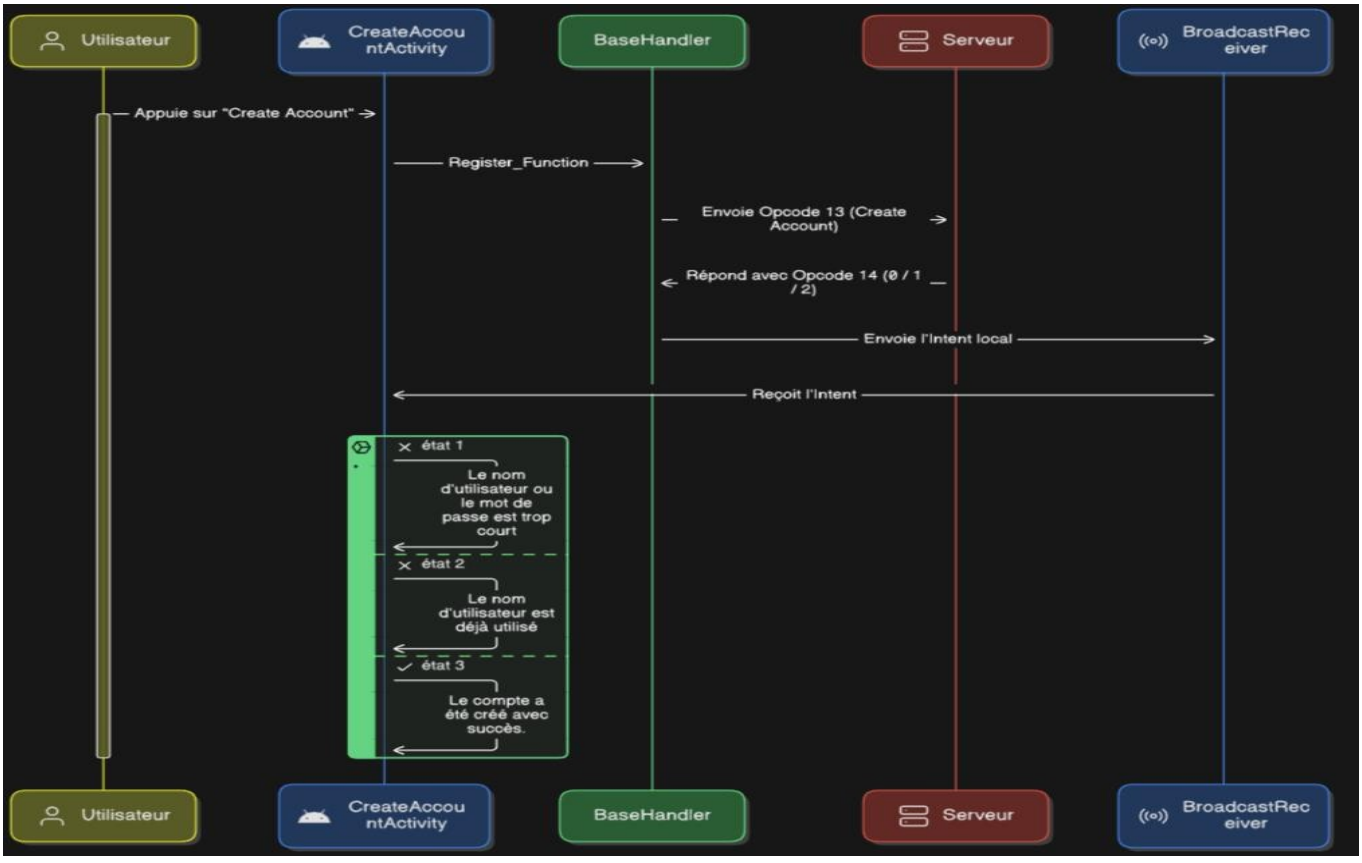


Figure 4.13 : Schéma de création de compte dans CreateAccountActivity.

4.3.5. RoomSetupFragment - Ajout d'une nouvelle pièce :

4.3.5.1. Fonctionnement général :

Le RoomSetupFragment permet à l'utilisateur de créer une nouvelle pièce dans le système domotique. Pour cela, l'utilisateur :

- Saisit le nom de la pièce.
- Génère automatiquement un code unique de 30 caractères.
- Sélectionne une icône représentant la pièce.
- Appuie sur le bouton « Add Room » pour envoyer la demande au serveur.

4.3.5.2. Données envoyées au serveur (Opcode 62) :

Lors de l'ajout, les données suivantes sont envoyées :

- **30 bytes :** Code de la pièce (généré automatiquement)
- **30 bytes :** Nom de la pièce (saisi par l'utilisateur)
- **4 bytes** : Identifiant de l'icône (sous forme de int)

4.3.5.3. Réponse du serveur (Opcode 63) :

Le serveur répond avec un seul octet :

- **0 :** échec la pièce existe déjà
- **1 :** succès la pièce a été ajoutée

Dans la classe BaseHandler, la méthode process_add_room_response traite cette réponse et l'envoie au fragment via un LocalBroadcast :

```
Intent intent = new Intent( action: "Room_Setup_Fragment");
intent.putExtra( name: "addRoomResponseStatus", (int) status);
LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Figure 4.14 : Déplacer la réponse vers RoomSetupFragment.

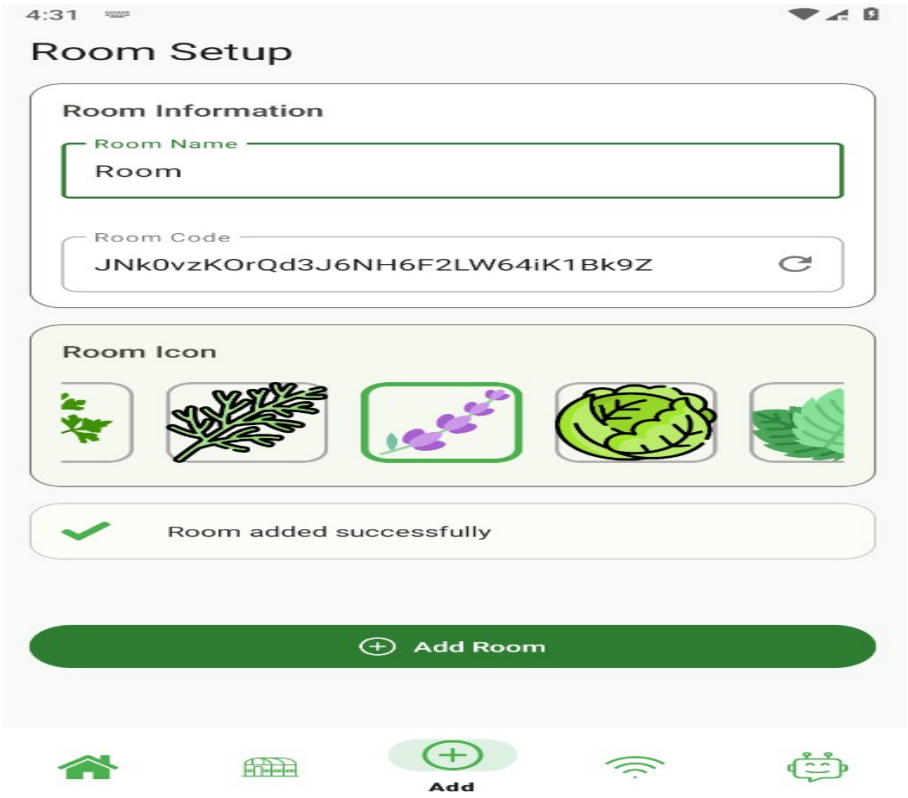


Figure 4.15 : RoomSetupFragment résultat (état 1).

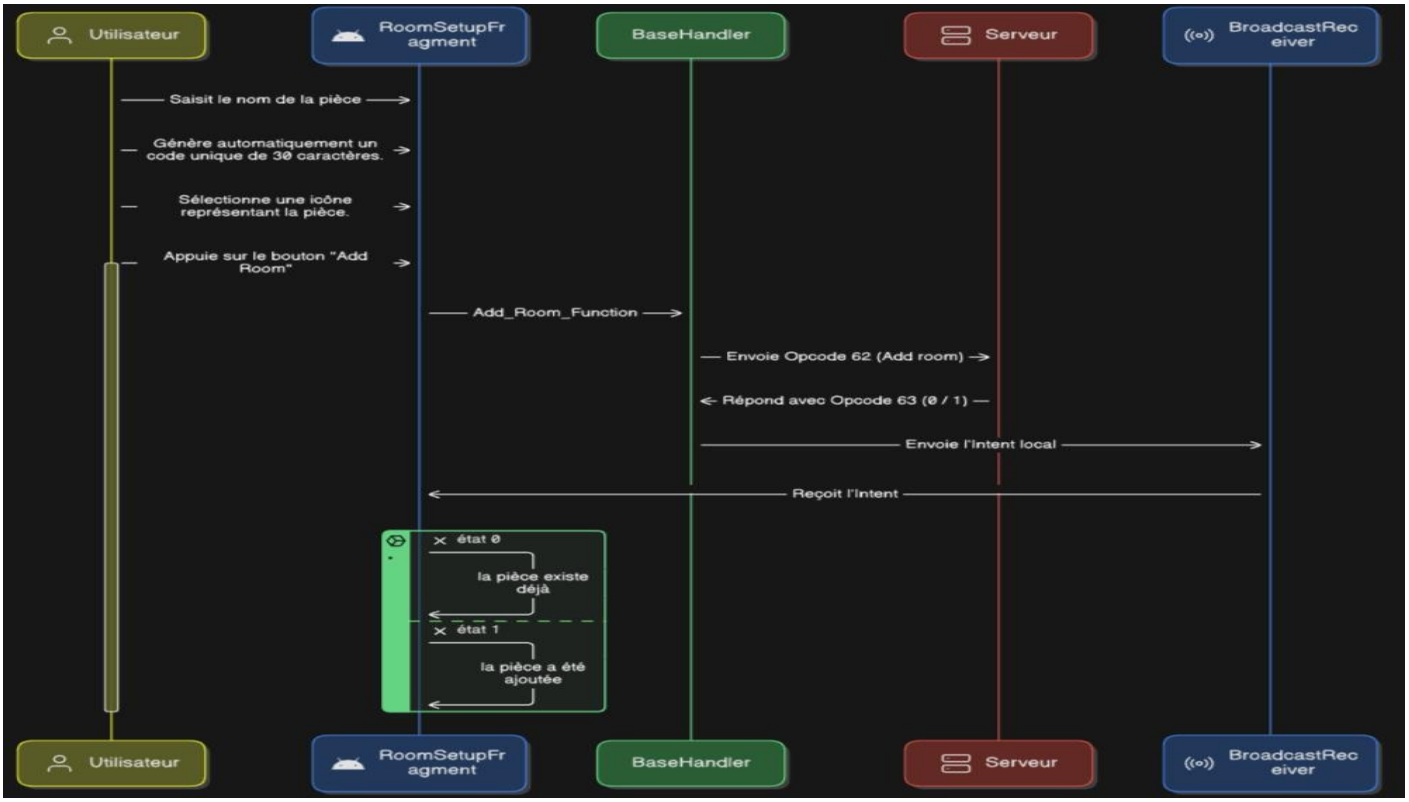


Figure 4.16 : Schéma pour l'ajout d'une pièce via RoomSetupFragment.

4.3.6. Ajout d'un appareil - AddDeviceFragment :

4.3.6.1. Fonctionnalité du Fragment :

Ce fragment permet à l'utilisateur d'ajouter un appareil à une pièce existante. Quatre étapes principales sont impliquées dans cette interface :

- Saisir le nom de l'appareil.
- Scanner le code de l'appareil (via un QR code).
- Sélectionner une pièce depuis une liste déroulante.
- Cliquer sur le bouton pour ajouter l'appareil.

4.3.6.2. Envoi de la requête au serveur :

Une fois ces informations remplies, l'application envoie une requête au serveur avec l'opcode 3, accompagnée des données suivantes :

- **30 octets :** code de la pièce.
- **30 octets :** nom de l'appareil.
- **30 octets :** code de l'appareil.

4.3.6.3. Réponse du serveur :

Le serveur répond avec l'opcode 4, contenant :

1 octet : Statut de la réponse

- 0 : l'appareil existe déjà.
- 1 : l'appareil a été ajouté avec succès.

Cette réponse est traitée dans la méthode process_add_device de la classe BaseHandler, qui émet ensuite un broadcast local :

```
Intent intent = new Intent( action: "Room_Add_Device");
intent.putExtra( name: "ADD_STATUS", (int) result);
LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Figure 4.17 : Déplacer la réponse vers AddDeviceFragment.

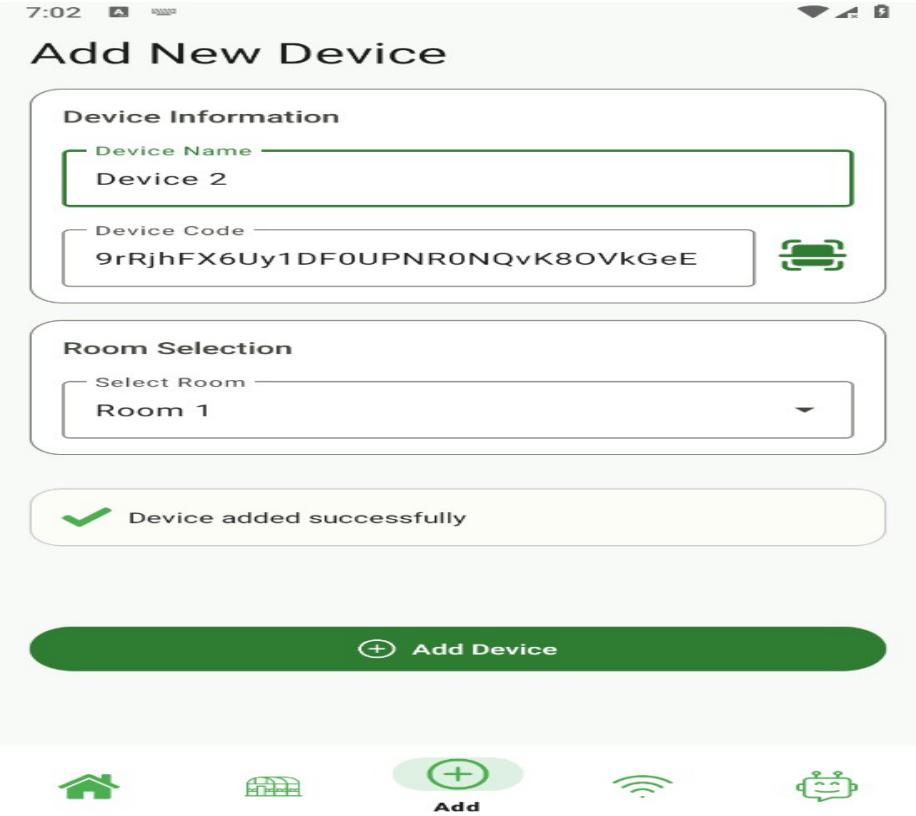


Figure 4.18 : AddDeviceFragment résultat (état 1).

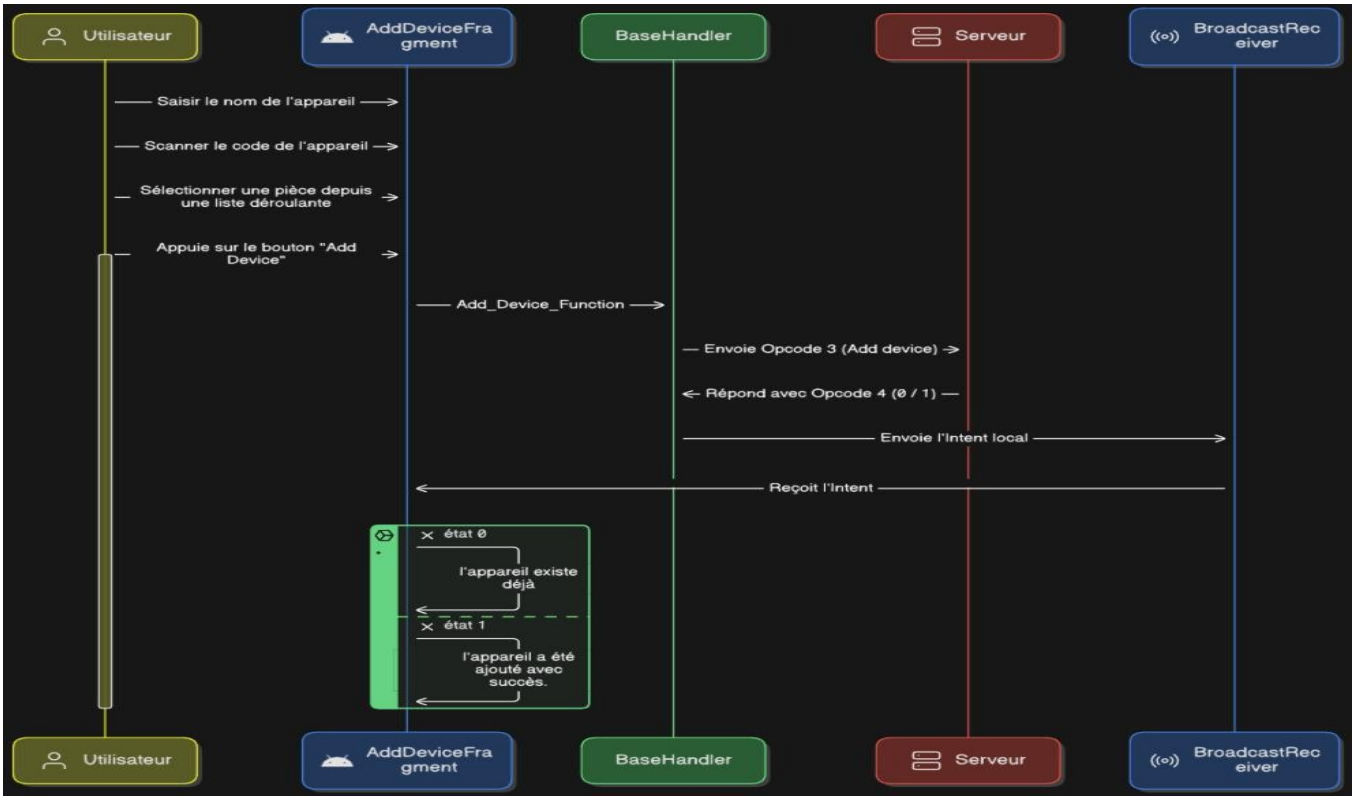


Figure 4.19 : Schéma pour l'ajout d'un appareil.

4.3.7. Fragment SensorDashboard :

Le fragment SensorDashboardFragment est responsable de l'affichage des données des capteurs en temps réel et historiques sous forme de graphiques, ainsi que du contrôle des actionneurs (relais). Il communique avec le serveur via plusieurs opcodes spécifiques.

4.3.7.1. Communication avec le serveur :

Réception des données en temps réel :

Lorsque le serveur envoie l'opcode 30, il transmet une trame contenant :

- **30 octets :** le code de l'appareil.
- **2 octets :** le type de données (ex. : opcode 30 pour valeur capteur).

Si l'opcode est 30 :

- **1 octet :** le type de capteur (température, humidité...).
- **4 octets (float)** : la valeur du capteur.

Si l'opcode est 40 :

- **1 octet** : type de relais (pompe, ventilateur...).
- **1 octet** : valeur du relais (0 = OFF, 1 = ON).

Ces informations sont envoyées à l'interface via :

```
Intent intent = new Intent( action: "SensorInfo");
intent.putExtra( name: "_DeviceCode", deviceCode);
intent.putExtra( name: "RealTimeType", (int) data_type);
intent.putExtra( name: "RealTimeValue", data_value);
```

Figure 4.20 : Déplacer les informations vers SensorDashboard fragment

Ou, pour le relais :

```
intent.putExtra( name: "DeviceCode", deviceCode);
intent.putExtra( name: "relay_type", (int)relay_type);
intent.putExtra( name: "relay_value", (int) relay_value);
```

Figure 4.21 : Envoi de l'état des relais au SensorDashboard fragment.

4.3.7.2. Données historiques - Opcode 70 :

Lorsqu'on appuie sur les boutons **Week** ou **Today**, un paquet est envoyé via Opcode 70.

Contenant :

- **30 octets** : code de l'appareil
- **1 short** : type de période (1 pour aujourd'hui, 7 pour semaine)

4.3.7.3. Réponse du serveur :

Les données arrivent dans la fonction process_data_info comme suit :

- **1 octet :** type de période (1 ou 7)
- **1 octet** : type de capteur
- **1 octet** : taille des données

  Boucle sur taille :
- **4 octets (float)** : valeur mesurée
- **4 octets (int)** : timestamp UNIX

4.3.7.4. Traitement dans le Fragment :

Lors de la réception de l'intent "SensorInfo" ou "AddInfo", on effectue :

- **Données en temps réel :**

```
soilTemperatureData.addData(RealTimeValue, currentTime);
```

En fonction du type, la donnée est ajoutée à la courbe correspondante.

- **Données historiques :**

```
soilTemperatureData.addMultiData(GlobalValues.soilTemperature_Values,
GlobalValues.soilTemperature_Times);
```

Les données sont ajoutées en masse à partir des listes globales.

4.3.7.5. État des relais :

On compare le code appareil reçu avec celui sélectionné, puis on met à jour la couleur du bouton selon l'état (vert = ON, rouge = OFF).

4.3.7.6. Contrôle des relais - Opcode 9 :

Lorsque l'utilisateur appuie sur un bouton de contrôle (**btn_pomp**, **btn_fan**, etc.), une trame est envoyée :

- **2 octets (short) :** type de relais (1 à 4)
- **1 octet** : état souhaité (0 ou 1)

Cela permet d'activer/désactiver la pompe, le ventilateur, la lumière ou le chauffage.

Composant	Type de relais (relay_type)
Pompe	1
Ventilateur	2
Lumière	3
Chauffage	4

Tableau 4.2 : Résumé des types de données.

Tableau 4.3 : SensorDashboardFragment résultat.

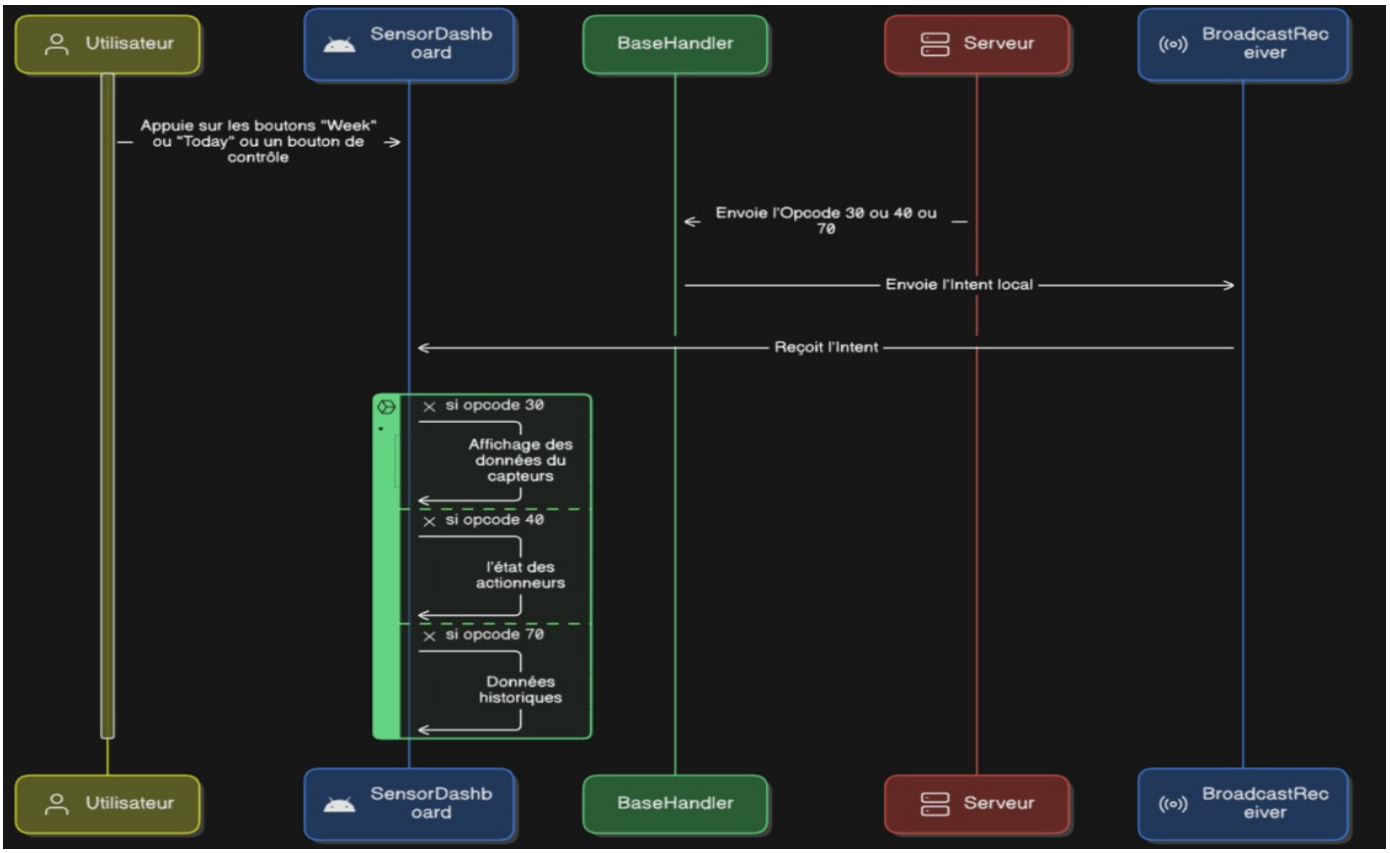


Figure 4.22 : Schéma de fonctionnement de la fragment SensorDashboard

4.4. Conception du boîtier externe et interne du projet :

Les images suivantes illustrent la conception du boîtier du projet, aussi bien à l'extérieur figure (4.23) qu'à l'intérieur (figure 4.24) .

Le design interne met en évidence l'agencement des composants (ESP32, batteries, module de charge TP4056, capteurs, etc.) de manière fonctionnelle et sécurisée.

Le design externe montre l'apparence générale du boîtier, les ouvertures prévues pour les capteurs, les connexions et la ventilation.

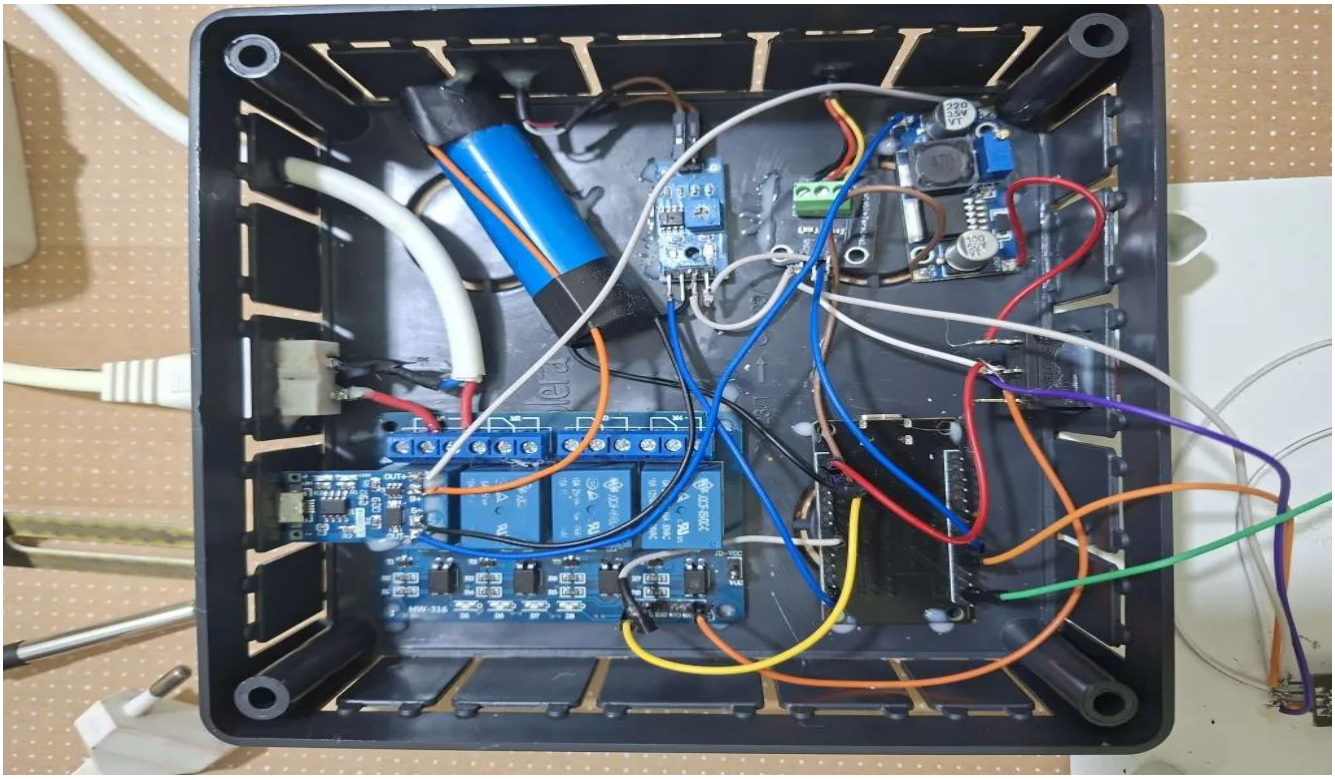


Figure 4.23 : Vue intérieure du boîtier.

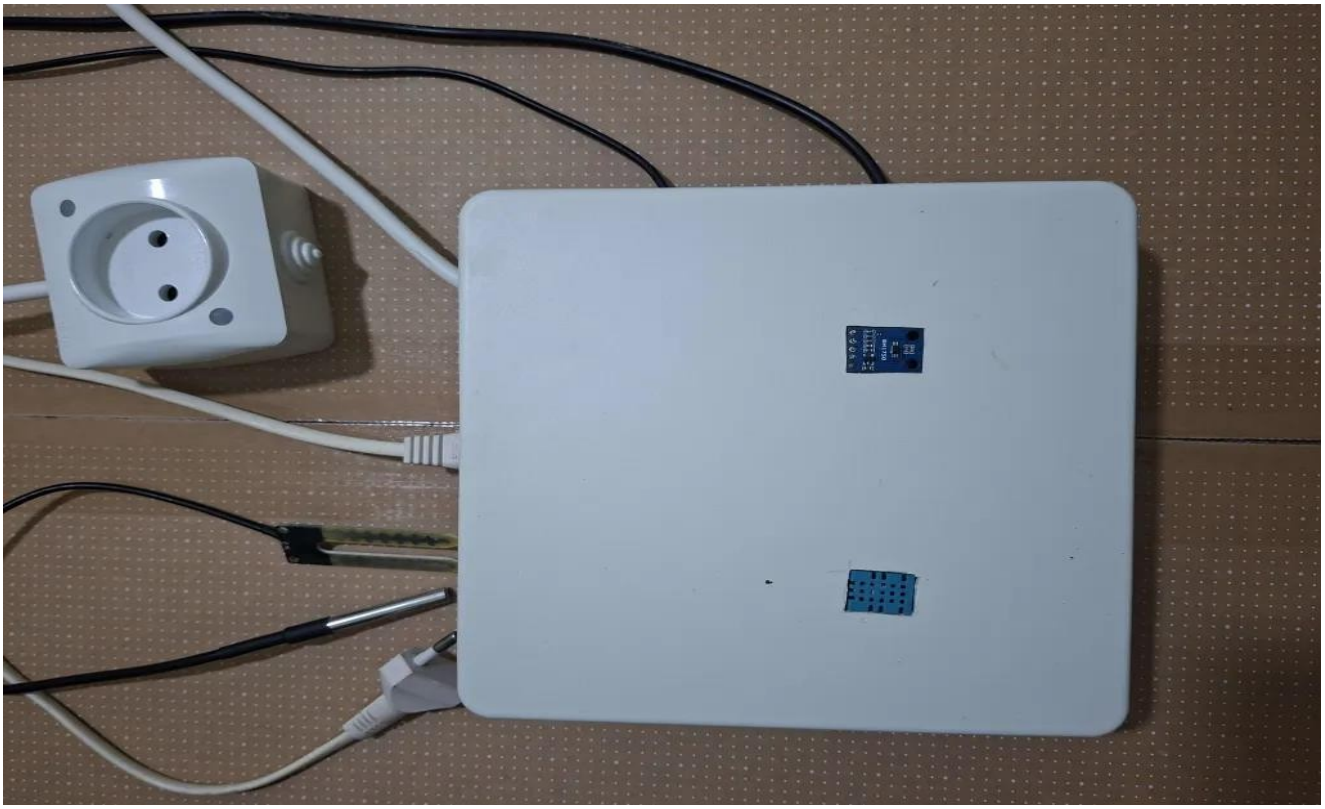


Figure 4.24 : Vue extérieure du boîtier.

4.5. Conclusion

Ce mécanisme de communication centralisé entre les Fragments et le serveur TCP garantit une architecture propre, modulable et facilement maintenable. Grâce à la classe BaseHandler et à l'utilisation du LocalBroadcastManager, chaque Fragment peut interagir de manière asynchrone avec le serveur tout en restant indépendant des détails de bas niveau liés à la gestion réseau.

Conclusion générale

En conclusion, le IOT se positionne comme une innovation majeure qui change la façon dont nous interagissons avec le monde. Leur intégration dans différents domaines, tels que l'agriculture et l'environnement, fournit des solutions intelligentes et efficaces pour relever les défis actuels, notamment ceux liés au changement climatique. Dans le contexte de la croissance des plantes, les technologies IoT permettent une surveillance précise et continue des paramètres environnementaux, garantissant des conditions optimales pour leur développement. Cela est particulièrement crucial face aux perturbations climatiques, où des outils intelligents peuvent anticiper les besoins des plantes et prévenir les risques liés aux variations extrêmes de température, d'humidité ou de qualité de l'air.

Références Bibliographiques

[1] Wikipedia, "Internet des objets", Wikipédia, [En ligne]. Disponible : https://fr.m.wikipedia.org/wiki/Ilnternet_des_objets. [Consulté:12 avril 2025].

[2] Mutualia, "Internet des objets (oT) pour une agriculture plus intelligente", Mutualia,f, [En ligne]. Disponible : https://www.mutualia.fr/lagriculteur/infos/economie-et-societe/news/internet-des= obiets-iot-pour-une-agriculture-plus-intelligente. [Consulté : 3 mai 2025].

[3] Digi, "L`IoT dans V'agriculture : une technologie au service du rendement", Digi International, [En ligne]. Disponible : https://fr.digi.com/blog/postiot-in-agriculture. [Consulté : 12 avril 2025].

[4] Mutualia, "Internet des objets (oT) pour une agriculture plus inteligente", Mutualia fr, [En ligne]. Disponible : https://www.mutualia.fr/lagriculteur/infos/economic-et-socicte/news/internet-des obiets-iot-pour-une-agriculture-plus-intelligente. [Consulté :8 juin 2025].

[5] RS Franee, "Les protocoles de communication IoT", RS Online, [En ligne]. Disponible: communication-iot. [Consulté : 3 juin 2025]:

[6] Power & Beyond, "How IoT is utilized", Power-and-Beyond.com, [En ligne]. Disponible : https://www.power-and-beyond.com/how-iot-is-utilized. [Consulté :3 juin 2025].

[7)] TotalEnergies, "Quelle est la difference entre courant alternatif (AC) et courant continu (DC)?", Charge+, [En ligne]. Disponible : https://chargeplus.totalenergies.com/fr/fag/courants-ac-dc. [Consulté :2 juin 2025).

[8] IP Systèmes, "Qu'est-ce qu un capteur connecté IoT ?", IP-Systemes.com, [En ligne]." Disponible : https://www.ip-systemes.com/quest-ce-quun-capteur-connect-iot.html. [Consulté : 10 mai 2025].