Rocher Ludovic

POLYTECH LYON

FILIÈRE INFORMATIQUE – 5A

Rapport Projet

IOT

Présentation de mon

Objet connecté

2023 – 2024

En raison de la croissance rapide des zones urbaines, les oiseaux ‘urbains’ sont confrontés à des défis croissants pour trouver des habitats adaptés et sûrs. Les variations climatiques, le manque de nichoirs appropriés, et l'impact de la pollution lumineuse peuvent affecter leur reproduction et leur survie. A cela il y a deux conséquences majeures :
• Le déclin des populations d’oiseaux en milieu urbain
• Menant alors à un déséquilibre écologique.

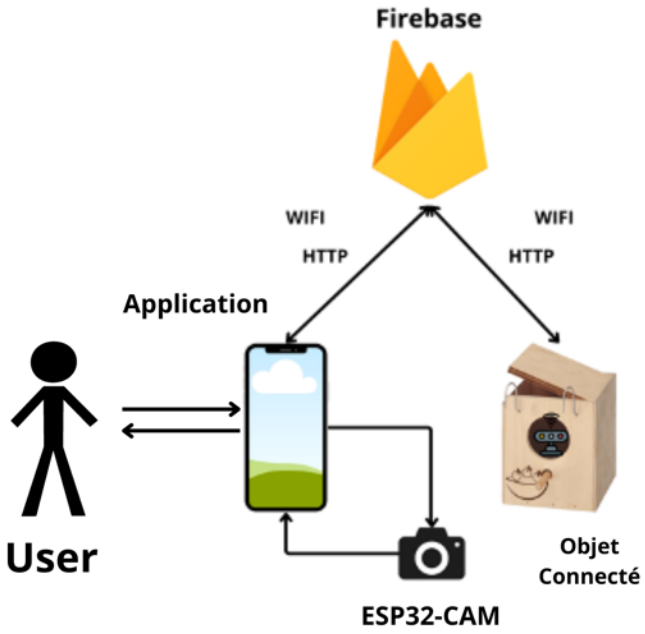
C’est pourquoi nous proposonts une solution : Un nid connecté offrant un environnement contrôlé aux oiseaux en milieu urbain. L’utilisateur pourra surveiller et réguler différents paramètres. L’objet comprendra une partie gérée par le microcontrôleur ESP32 et une application mobile en lien avec ce-dernier.

Le projet de Nid Connecté vise à créer un système permettant aux utilisateurs de surveiller et d'interagir avec un nid à distance. On utilisera pour cela l’application mobile *Nestrack*.

Concrètement, notre nid connecté permettra d’offrir la surveillance en temps réel des conditions environnementales à l’aide d’une application mobile sécurisée. L’utilisateur pourra visualiser différents paramètres physiques (Température, humidité, luminosité et niveau sonore). Il y a également une caméra intégrée qui permet de visualiser en direct l’intérieur du nid ainsi qu’un système de Leds et de chauffage que l’utilisateur pourra allumer ou éteindre à sa guise.

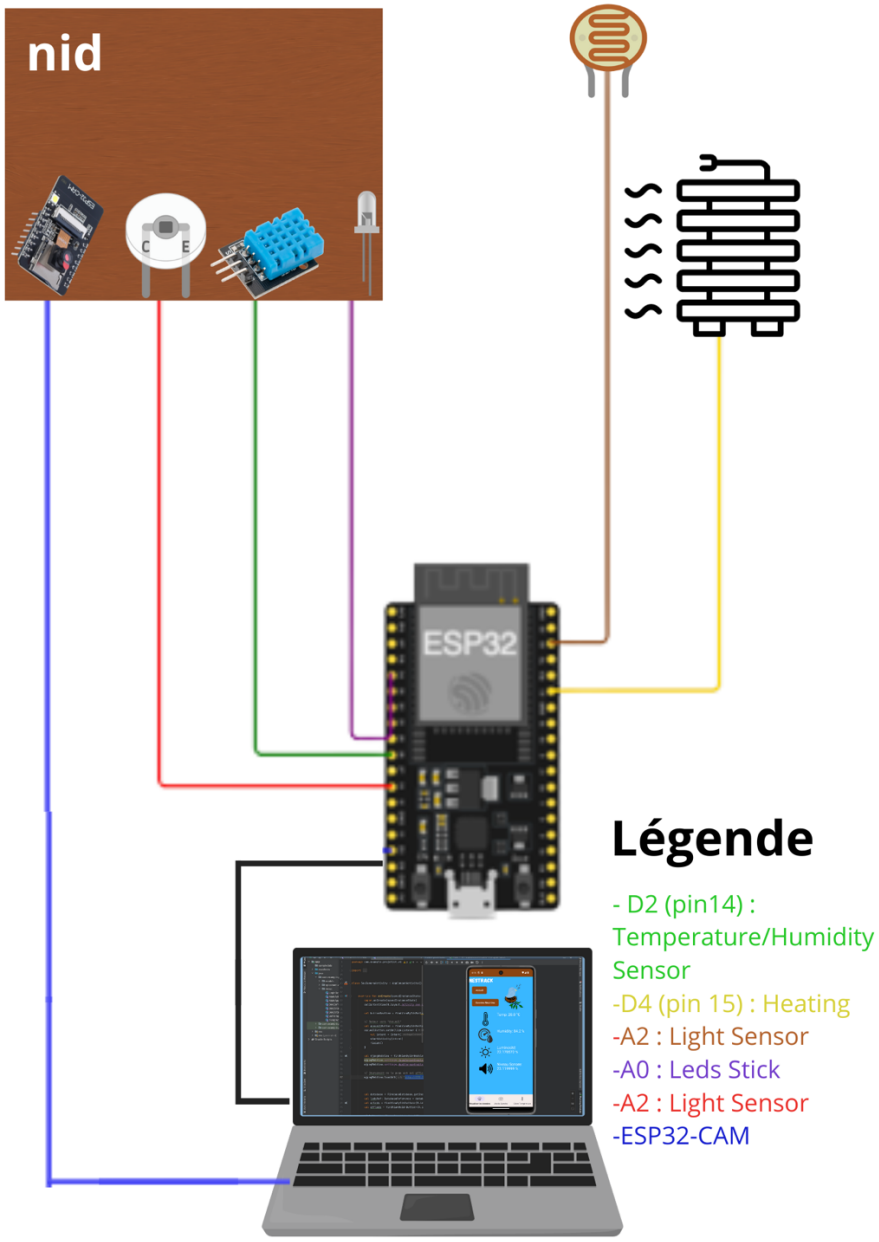
Voici la liste des technologies qui seront utilisées pour ce projet :
• L’IDE Arduino afin de programmer notre microcontrôleur et gérer la collecte de données ainsi que la gestion des capteurs et Controller. Nous connecterons notre carte à un réseau Wi-Fi afin de permettre la connexion avec une API.
• Android Studio avec Kotlin pour le développement de notre application mobile. Cela permettra à l’utilisateur d’observer en temps réel et contrôler les capteurs directement via l’appli en interagissant également avec notre API.
• Afin de permettre le lien entre Arduino et notre application mobile, nous mettrons en place une connexion WiFi via l’API Firebase Realtime Database.

Pour résumer, voici un schéma de déploiement permettant de comprendre le fonctionnement de ce projet :



Pour se faire, voici la liste du matériel qui sera utilisé :
• Un nid d'Oiseau (en bois) pour extérieur qui sera l'objet principal dans lequel nous placerons nos capteurs.
• Le microcontrôleur ESP32 Adafruit feather : Il servira de cerveau pour notre système embarqué, gérant la collecte de données à partir des capteurs (Température/humidité, sonore et photorésistance) ainsi que le contrôle des actionneurs.
• Un capteur de Température/Humidité qui sera utilisé pour surveiller les conditions à l'intérieur du nid.
• Un capteur de luminosité qui servira à mesurer la luminosité ambiante autour du nid.
• Un capteur Sonore qui détectera les sons émis par les oiseaux dans le nid.
• Une carte ESP32-CAM permettant l'intégration d'une caméra pour surveiller l'intérieur du nid, fournissant un flux vidéo en directe.
• Des LEDs qui seront utilisées pour contrôler l'éclairage du nid.
• Un chauffage lié à une multiprise que l'on pourra activer/désactiver pour que le chauffage puisse s'activer.

Le montage de l'objet connecté sera le suivant :



Pour l'utilisation

Côté Arduino

L'utilisateur devra configurer une connexion WiFi ainsi qu'à Firebase afin de communiquer les données de l'ESP32 vers notre application et inversement (pour les controllers). Il n'aura alors qu'à activer le microcontrôleur pour avoir son objet connecté. (Une version commercialisable comprendrait un microcontrôleur WiFi directement relié à un réseau, sans avoir à l'alimenter avec l'ordinateur, c'est ici un prototype que l'on propose).

Côté Android Studio

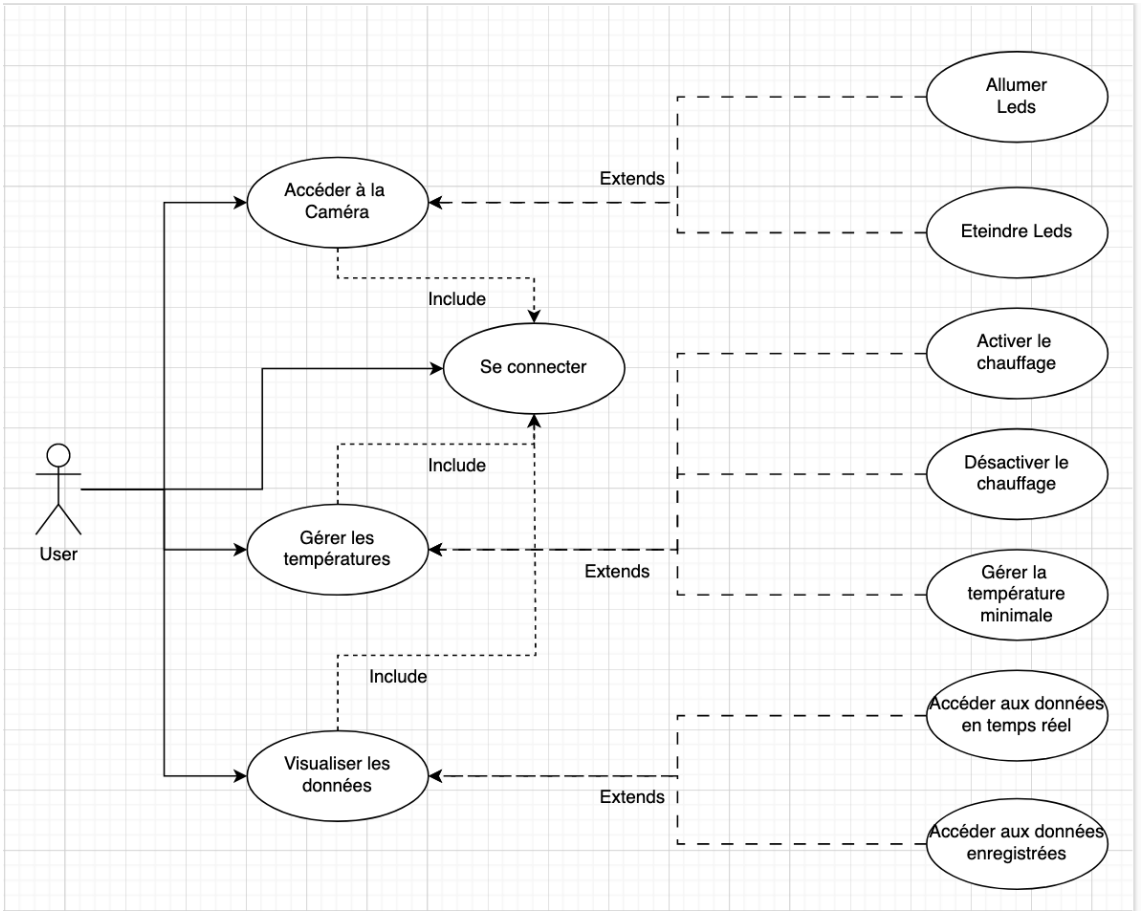
Lorsque l'utilisateur lance l'application, il a un court chargement le faisant arriver à la page d'accueil. Il doit alors se connecter (les identifiants et mot de passe sont enregistrés sur Firebase) pour accéder à l'application. Une fois sur la page principale, l'utilisateur a accès à 3 modules :

• L'accès aux données en directes et récoltées. L'utilisateur verra en temps réel la température, humidité, niveau sonore et luminosité s'afficher sur son application. Il pourra choisir d'aller vers les données récoltées qui permet d'afficher les moyennes valeurs max et min des valeurs enregistrées pour ces différents paramètres.

• Le module de caméra. L'utilisateur obtient le flux vidéo en temps réel. Comme l'intérieur du nid peut être sombre (notamment hors journée ou selon le modèle du nid), il pourra choisir d'activer ou désactiver les Leds à distance sur cet écran.

• La gestion des températures. L'utilisateur pet choisir une température minimale qui sera comparée à la température actuelle. En cas de basse température, il peut activer/désactiver le chauffage.

Voici un diagramme de cas d'utilisation permettant de résumer les différents modules et les actions que peut faire l'utilisateur depuis l'application mobile :



Pour le développement

Côté Arduino

Pour la gestion des capteurs et des Controller, j'ai utilisé l'IDE Arduino.
J'ai développé un *Main.ino* étant le fichier que l'on utilise pour contrôler toute la partie technique de l'objet ainsi que la connexion WiFi et à Firebase. Afin de rendre le code plus clair et pour un contrôle plus simple, j'ai choisi d'utiliser des fichiers *.h* et des *.cpp* pour les différents capteurs et controllers. Cela correspond à une pratique courante en POO. Ainsi cela me permet de séparer la déclaration des classes, des fonctions et des variables de leur implémentation.
Ainsi pour chaque capteur et controller j'ai un fichier *.h* qui lui est propre et qui définit sa classe (par exemple DHTSensor.h pour le capteur de température et humidité) et un fichier *.cpp* regroupant les fonctions et variables de ce capteur (respectivement DHTSensor.cpp). Je peux alors les initialiser et les implémenter directement dans le Main ensuite.

Pour la connexion à la WiFi, j'entre les paramètres SSID et le mot de passe du réseau WiFi puis je lance la connexion.
Une fois la connexion WiFi établit, je lance une connexion Firebase en initialisant l'hôte (le lien de ma base de données en temps réel), la clé API et la clé secrète pour établir le lien avec la base de données Firebase.
Une fois la connexion à Firebase établit, la fonction loop() se lance. Les capteurs envoient alors à intervalle de 5 secondes les données récupérées pour les stocker sur la base de données en temps réel Firebase. On va aussi lire booléens pour savoir si l'on doit activer/désactiver les Leds ou le chauffage.

Pour permettre d'afficher la webcam, j'utilise aussi une ESP32-CAM. Pour son contrôle, on sélectionne la carte AI THINKER ESP32-CAM puis l'on prend l'exemple :
ESP32 -> Camera -> CameraWebServer
On entre les paramètres WiFi. Après manipulation et téléversement, on obtient un lien nous permettant d'accéder au live de la caméra. Pour ma part, pour simplifier son utilisation, j'ai ajouté les lignes suivantes pour garder le même lien :

```
IPAddress staticIP(172, 20, 10, 3);
IPAddress gateway(172, 20, 10, 1);
IPAddress subnet(255, 255, 255, 0);
```

Ainsi le lien utilisé côté mobile reste le *'http://172.20.10.3.81/stream'*.

Voici la liste des librairies Arduino que j'ai utilisera dans notre projet :

Arduino
WiFi.h pour la gestion de connexion et communication avec le réseau Wi-Fi
WifiClient.h pour la communication client-serveur sur le réseau WiFi
Firebase.h permettant le lien entre Arduino et Firebase, offrant une base de données en temps réel
DHT.h pour l'interaction avec le capteur de température et humidité
Adrafruit_NeoPixel.h pour le contrôle des leds
esp_camera.h pour utiliser la caméra

Côté Android Studio

Passons maintenant au développement de l'application.

Le développement de l'application a été fait à l'aide de Android Studio en Kotlin. J'ai développé mon application avec des Vues qui correspondent aux Activity permettant de contrôler ce qui se passe sur les différentes pages de notre application. Ces Activity sont associées à des layout (fichiers xml) qui permettent de gérer le visuel de nos pages. Les différentes Activity permettent de naviguer sur l'application en lançant d'autre Activity, elle permet les requêtes envers Firebase pour récupérer les données stockées mais aussi d'envoyer des données vers Firebase (pour les controllers). Le visuel de notre caméra se fait également via une page web alimentée par notre ESP32-CAM que l'on charge en flux MJPEG.
J'ai aussi des modèles '*SensorData'* qui est une classe de données pour les données relevées par les capteurs et '*DataCalcul'* qui définit les fonctions de calcul tels que la moyenne, la valeur minimale et maximale pour l'affichage des données sur le long terme.
Finalement, on a aussi des splash Activity qui correspondent aux écrans de chargement entre certaines Activity, auxquelles j'ai intégré une image GIF.

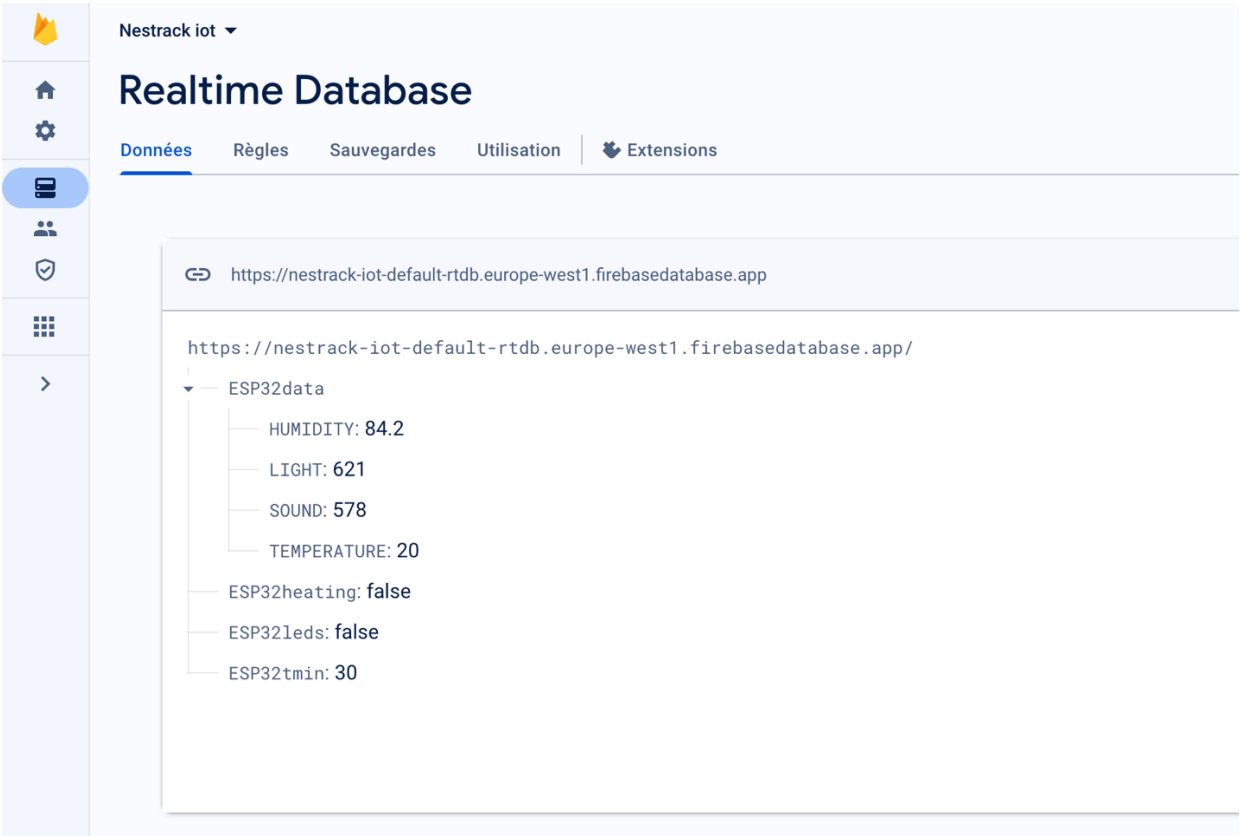
L'intégration de le la page web aura été l'étape la plus difficile. En effet l'ESP32-CAM est très sensible et les problèmes de communication sont assez fréquents. Une fois l'ESP32-CAM fonctionnel il m'a été aussi difficile de parvenir à afficher la vidéo en live. En effet, il y a très peu de documentation sur l'ESP32-CAM expliquant comment afficher convenablement le flux vidéo sur notre application. J'ai donc essayé de nombreuses solutions (plusieurs librairies d'intégration vidéo). C'est finalement avec la librairie intégrée *webkit.WebView* que j'ai pu afficher mon flux vidéo.

Voici la liste des dépendances Android Studio que l'on utilisera dans ce projet :

Android Studio
androidx.core pour des fonctionnalités du framework Android
androidx.appcompat pour la compatibilité descendante avec les anciennes versions d'Android
com.google.android.material fournissant des composants d'interface utilisateur
androidx.constrainlayout pour les mises en page complexes
pl.droidsonroids.gif pour afficher des images GIF
*com.google.firebase.** pour tous les services Firebase
junit pour les tests unitaires JUnit

Documentation de l'API

Firebase est une plateforme de développement d'applications mobiles et web proposée par Google offrant un ensemble de services cloud, d'outils et de bibliothèques qui facilitent le développement des applications. Parmi les services proposés par Firebase, la Firebase Realtime Database est un service de base de données NoSQL en temps réel hébergé par le cloud et synchronisant les données en temps réel. C'est donc cette fonctionnalité que j'ai utilisé en tant qu'API afin de faire le lien entre l'ESP32 et mon application.



Nous avons donc les nœuds suivants :

1. **ESP32data/** (un nœud contenant les informations suivantes :
   • **TEMPERATURE** qui stocke la valeur de la température en direct
   • **HUMIDITY** qui stocke la valeur de l'humidité en direct
   • **SOUND** qui stocke la valeur du niveau sonore en direct
   • **LIGHT** qui stocke la valeur de la luminosité en direct

2. **ESP32light** pour gérer, à l'aide d'un booléen, le système on/off des Leds

3. **ESP32heating** pour gérer, à l'aide d'un booléen, le système on/off du chauffage

4. **ESP32tmin** pour stocker la valeur de température minimale choisie par l'utilisateur sur l'application.

Bilan et Conclusion :

Le projet de Nid Connecté est conçu de sorte à répondre à la problématique initiale avec un système de surveillance à distance d'un nid d'oiseau. La sélection du matériel, comprenant l'ESP32, divers capteurs, une caméra, des LEDs, et un système de chauffage est pertinente pour les objectifs du projet. L'utilisation d'Arduino pour la programmation de l'ESP32, ainsi que d'Android Studio avec Kotlin pour le développement de l'application mobile m'ont apporté de solides compétences sur ces derniers.

La structure du code Arduino, avec des fichiers .h et .cpp séparés pour chaque capteur et actionneur est une approche favorisant la modularité et la lisibilité du code.
La connexion à Firebase Realtime Database pour le stockage et le partage des données entre l'ESP32 et l'application mobile est une solution efficace permettant un stockage des données en temps réel.
Le développement de l'application Android avec des vues (layout) correspondant aux différentes activités et des modèles pour la gestion des données est également une solution efficace pour le développement de l'application.

A l'avenir, voici quelques améliorations pour l'objet connecté :

• Un stock des données sur le long terme permettant d'afficher (et éventuellement tracer) les variations des paramètres captées sur de longues périodes, et permettant d'afficher les valeurs moyennes, max et min sur de longues périodes également.

• La possibilité pour l'utilisateur de prendre des photos avec la caméra qui pourront directement être enregistrées sur le téléphone.

• Un système de notification si la température minimale est atteinte afin d'avertir directement l'utilisateur et lui permettre d'activer le chauffage, ou bien si le niveau sonore est trop anormalement élevé.

• Un système de chauffage améliorée s'activant automatiquement lorsque la température atteinte est trop faible et qui s'arrête lorsque l'on a atteint un seuil assez stable.

Dans l'ensemble, le projet de Nid Connecté atteint ses objectifs en créant un système complet de surveillance à distance d'un nid d'oiseau et respectant les consignes de l'exercice.
Le Nid Connecté représente un travail mettant en œuvre une solution IoT complète pour la surveillance à distance, tout en utilisation une étude technique dans le domaine des objets connectés et du développement d'applications mobiles.