POLYTECH
LYON

IoT

RAPPORT DE PROJET

BENALI Myriam – NAAJI Dorian
POLYTECH LYON
5A INFO GROUPE 2, ÉQUIPAGE 2

INTRODUCTION

Ce présent rapport décrit le travail apporté pour la réalisation du projet d'IoT & Systèmes Embarqués. Nous avons eu pour objectif de produire une API, concevoir et programmer l'objet physique ainsi que la réalisation d'une application mobile.

1. PHASE 1 : CONCEPTION DU PROJET

1.1. Contexte du projet

Notre projet consiste en **la gestion d'un système d'éclairage d'une pièce** lors de la venue d'une personne (entrée d'une personne dans la pièce d'un appartement, d'un bureau, d'une chambre, etc.).

L'utilisateur signale son arrivée à l'entrée de la porte grâce au **lecteur NFC** via un badge/son téléphone. L'utilisateur rentre dans pièce et les **LEDs** s'allument car le **détecteur de mouvement** détecte la présence d'une personne dans la pièce. Cependant, **si la personne n'a pas badgé au préalable, les LEDs se s'allument pas**.

1.2. Le matériel nécessaire

Pour mener à bien ce projet, nous utiliserons un module microcontrôleur Wifi et Bluetooth, des capteurs et actionneurs :

- Microcontrôleur : **Starter Kit ESP32**,
- Capteurs : **lecteur NFC** et **capteur PIR**,
- Actionneurs : **quelques LED** ou un **mini-ruban LED**.

1.3. Les protocoles de communication

Nous utiliserons pour notre projet des communications HTTP entre les différents *devices*. Le téléphone communiquera directement avec le microcontrôleur, qui disposera d'une API REST permettant de communiquer avec le monde extérieur via le protocole HTTP.

1.4. Les diagrammes et l'architecture du projet

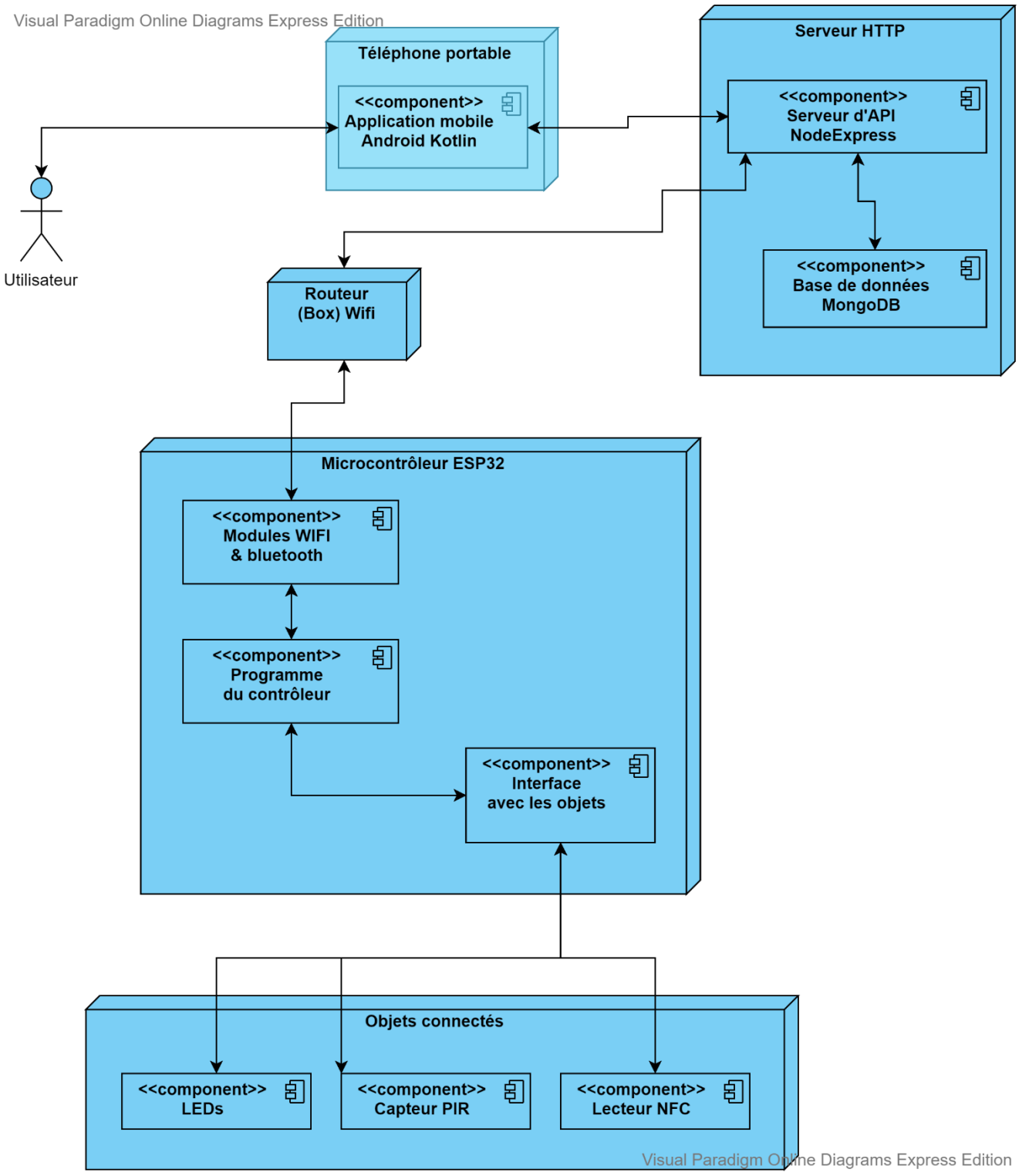


Figure 1 : Diagramme de déploiement

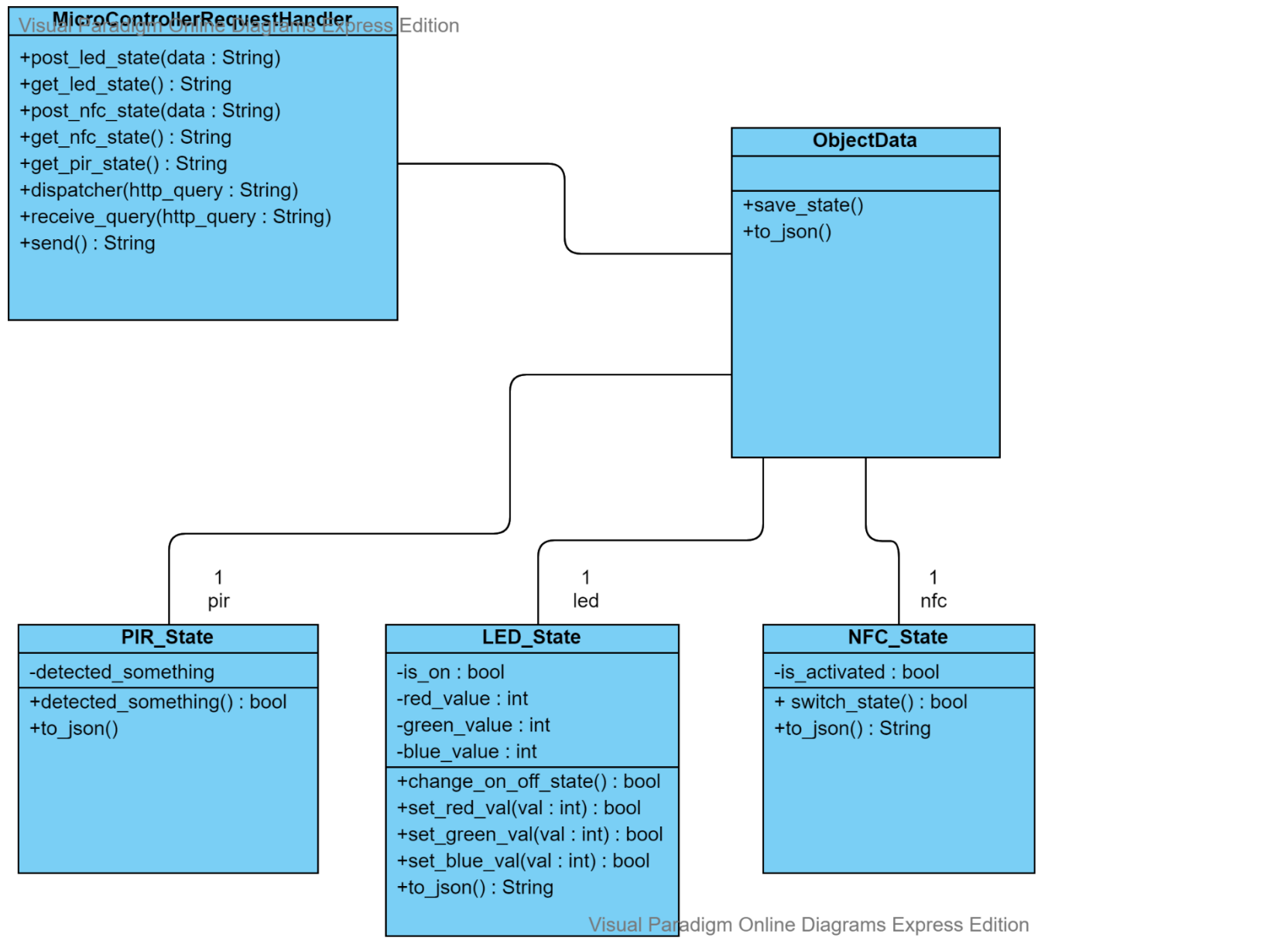


Figure 3 : Diagramme de classes

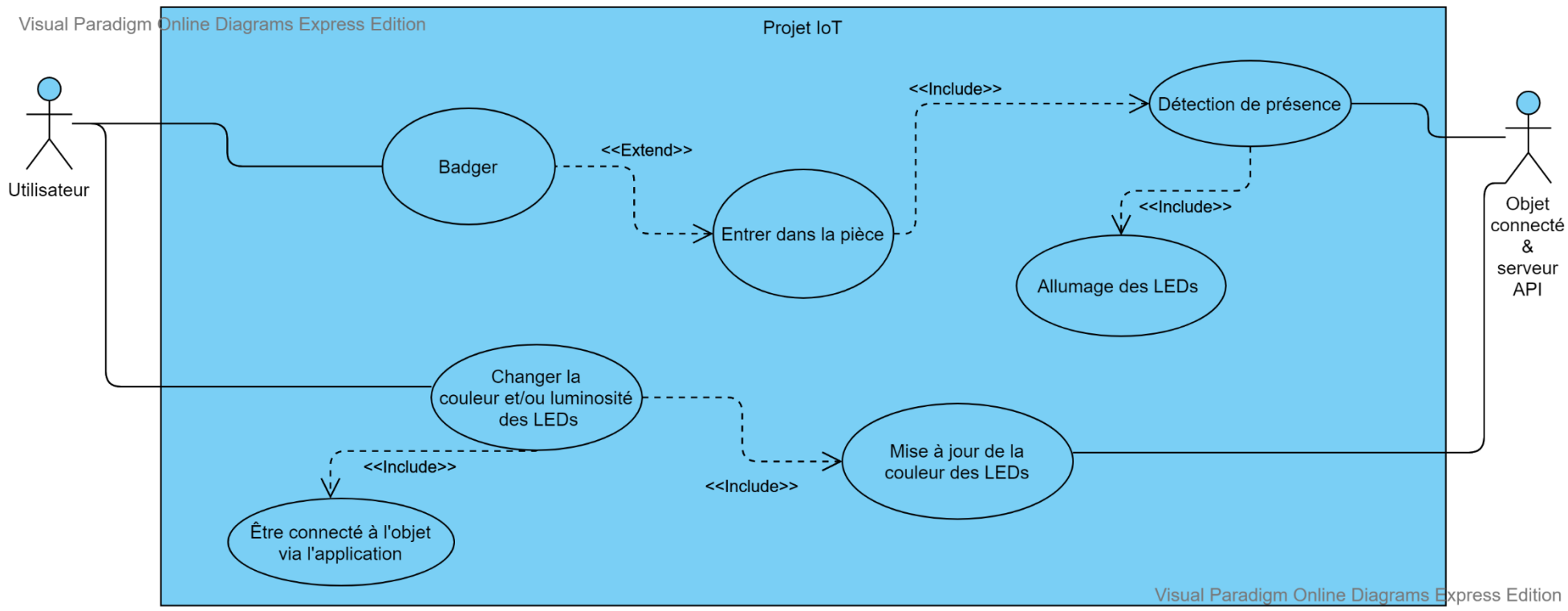


Figure 2 : Diagramme de cas d'utilisation

1.5. Maquettes de l'application

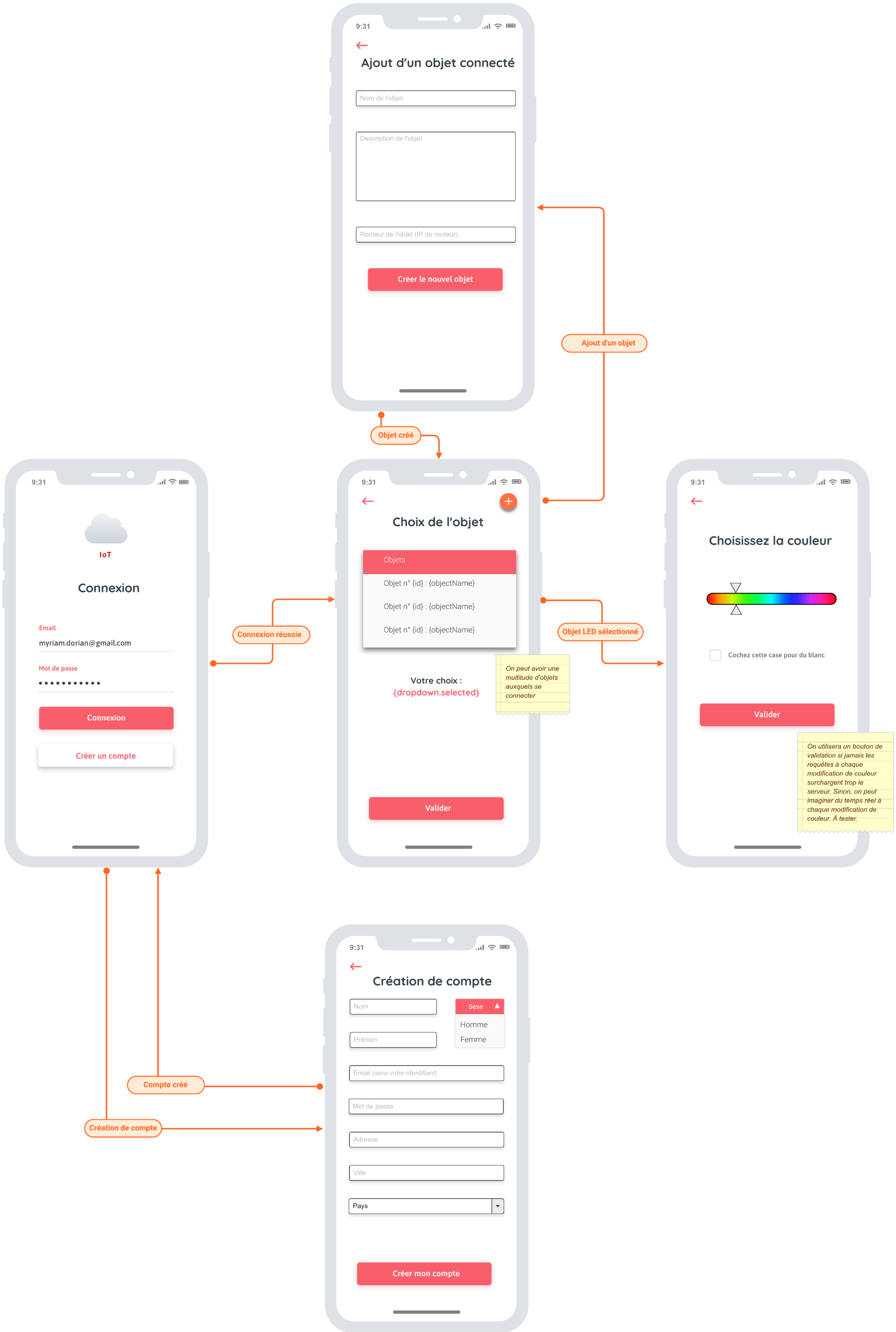


Figure 4 : Wireframe de l'application

1.6. Informations annexes

- Les LEDs ne s'allument que lorsque les deux conditions suivantes sont réunies :
  - L'utilisateur a badgé,
  - L'utilisateur est entré dans la pièce et le détecteur de mouvement a détecté une activité.

- L'application permettra de contrôler (en étant connecté au serveur) :
  - La couleur des LEDs
  - L'état des LEDs (allumé/éteint)
  - Leur luminosité *(à vérifier)*

- Les LEDs restent dans le même état entre chaque passage de badge (persistance des données sur l'API).

- Même si l'utilisateur ne bouge plus, et que le capteur de présence ne détecte rien, les LEDs restent allumées. Le capteur de présence sert juste après passage du badge à s'assurer que quelqu'un est bien présent dans la pièce.

- Si l'utilisateur badge lorsque les LEDs sont allumés, cela correspond à un départ et les LEDs s'éteignent. Les informations sont envoyées au serveur.

1.7. Gestion du projet

Pour la gestion du projet, nous utiliserons Trello, un tableau partagé permettant de regrouper l'ensemble des tâches devant être effectuées pour mener à bien le projet. Nous fonctionnons avec une méthodologie agile dans le sens où nous réalisons :

- des réunions régulières,
- un suivi permanent du projet,
- un partage des tâches optimisé.

2. PHASE 2 : DÉFINITION DE L'API DES WEB SERVICES & DOCUMENTATION DE L'API

2.1. Définition des routes de l'API

L'ensemble des routes ont été documentées et conçues sous l'outil Postman.

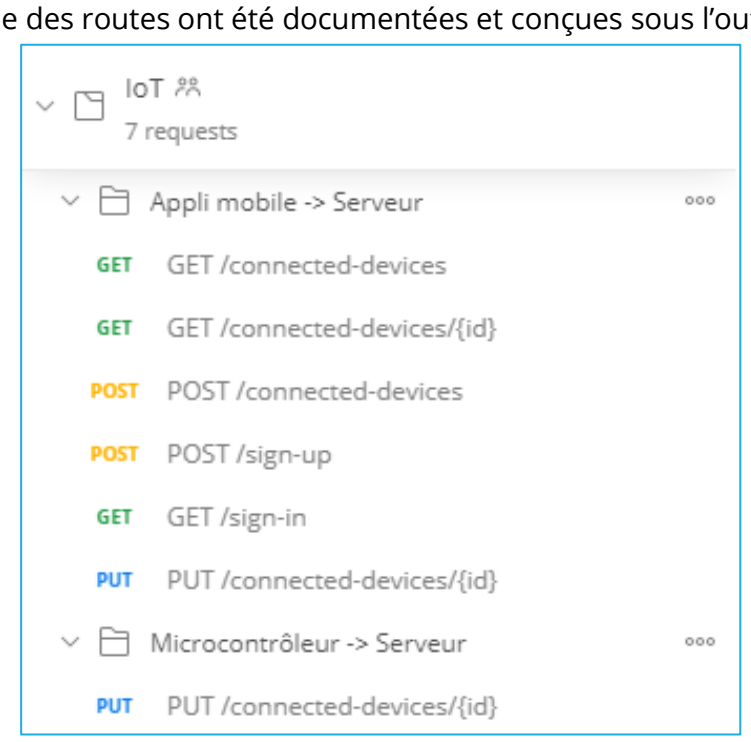


Figure 5 : Routes du serveur

L'application mobile aura ainsi accès à 6 routes :

- *GET /connected-devices* : Cette route permet à un utilisateur de récupérer l'ensemble des objets connectés gérés par le serveur. Cela permettra d'avoir sur l'application mobile un affichage des différents objets connectés gérés par le serveur.

- *GET /connected*-devices*/{id}* : Cette route permet à un utilisateur de récupérer un objet connecté selon son id. Les informées de l'objet connecté sont ensuite retournées.

- POST /connected-devices : Cette route permet à un utilisateur d'ajouter un nouvel objet connecté qui pourra être géré par l'API.

- POST /sign-up : Cette route permet à un utilisateur de créer un nouveau compte.

- GET /sign-in : Cette route permet à un utilisateur de s'authentifier avec un compte existant.

- PUT /connected-devices/{id} : Cette route permet à l'utilisateur de mettre à jour l'état d'un objet connecté. Les données seront changées dans un premier temps au niveau du microcontrôleur, qui mettra l'objet connecté à jour. Le microcontrôleur appellera ensuite à son tour la route appropriée pour mettre à jour les données en BD.

Le microcontrôleur lui aura accès à une seule route pour l'instant, qui lui permettra de mettre à jour l'état de(s) (l')objet(s) connecté(s) au(x)quel(s) il est relié :

- PUT /connected-devices/{id} : Cette route permet au microcontrôleur ESP32 (ou éventuellement un autre, l'architecture n'est pas figée) de mettre à jour l'état d'un objet connecté donné.

2.2. Format des données d'échange

Pour nos données d'échange, nous avons deux principaux objets qui seront utilisés dans le cadre de ce projet IoT. Bien sûr, l'API pourrait également traiter d'autres objets json si l'on ajoute de nouveaux objets connectés.

Un premier objet qui sera utilisé est l'*user*. Un exemple de sa structure JSON est la suivante :


```
{
    "email": "myriam.b@gmail.com",
    "password": "blabla",
    "firstName": "Myriam",
    "lastName": "B",
    "sex": true,
    "age": 23,
    "address": "rue bloblo",
    "city": "Villeurbanne",
    "country": "France"
}
```


Figure 6 : Exemple d'un objet *user*

La structure détaillée d'un *user* (détaillée ci-contre en figure 7) est également disponible de manière interactive en ligne à cette adresse : https://iot-polytechlyon.github.io/user-schema.html

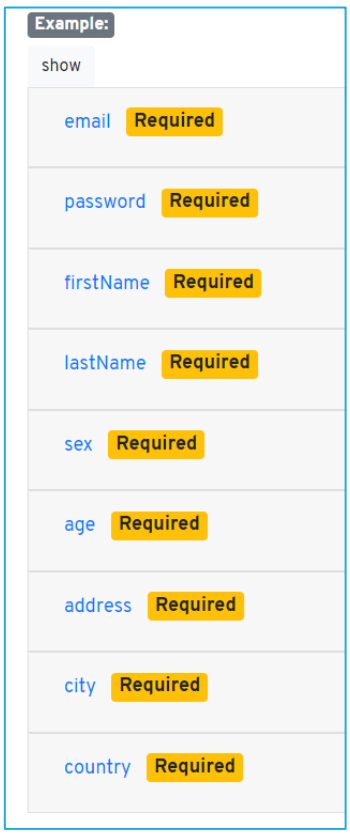


Figure 7 : Structure JSON d'un *user*

Un autre objet important que nous utiliserons est l'objet *connectedDevice*, dont voici un exemple ci-après :


```
{
    "name": "device_esp32_leds",
    "description": "that device links one esp32 microCPU with 1 NFC reader, 1 motion sensor and 3 LEDs.",
    "router": "192.168.0.19",
    "state": {
        "pir_state": {
            "detected_something": false
        },
        "nfc_state": {
            "is_activated": false
        },
        "led_state": {
            "is_on": false,
            "red_value": 144,
            "green_value": 17,
            "blue_value": 232
        }
    }
}
```


Figure 8 : Exemple d'un objet *connectedDevice*

L'objet *state* contenu au sein d'un *connectedDevice* est modulable et dépend de l'objet connecté. Ici, l'objet state décrit l'état de notre objet connecté spécifique au cadre de notre projet : un microcontrôleur ESP32 lié à un capteur PIR, à un lecteur NFC et à des LEDs.

Il est tout à fait envisageable (le modèle de l'API le permet) d'utiliser un objet *state* différent, propre à un autre objet connecté. Tout comme pour l'*user*, la visualisation interactive de la structure JSON d'un *connectedDevice* est disponible en ligne, ici : https://iot-polytechlyon.github.io/connectedDevice-schema.html

3. PHASE 3 : MISE EN PLACE DU SERVEUR D'API ET DE L'APPLICATION MOBILE

3.1. Mise en place du serveur

3.1.1. Technologies utilisées

Pour mettre en place le serveur de l'API, nous utiliserons :

- le langage JavaScript,
- node.js et express.js
- MongoDB

Le code source du serveur est disponible au sein du *repository GitHub* suivant :

https://github.com/IoT-PolytechLyon/iot-2a-server

3.2. Développement de l'application mobile

3.2.1. Visuel de l'application

Pour le développement de l'application mobile, nous nous tiendrons aux maquettes réalisées durant la phase 1 du projet. Elles sont également disponible sur moqups.com : https://app.moqups.com/3cfPlxezOO/view/page/ae8fe8eb0

3.2.2. Technologies utilisées

Pour l'application mobile, nous avons utilisé le langage Kotlin, qui permet de développer des applications Android sous Android Studio.

3.2.3. Communication avec le serveur

L'application mobile communique avec le serveur HTTP NodeExpress via un routeur. Nous utilisons la bibliothèque Kotlin Retrofit afin de réaliser des appels HTTP.

4. PHASE 4 : SYSTÈME EMBARQUÉ

4.1. Familiarisation avec le matériel

Pour ce projet, nous avons utilisé :

- Un microcontrôleur ESP-32 et un Shield Grove permettant de connecter différents éléments en *plug n play*. Parmi ces éléments figurent :
    - Un Grove NFC Reader
    - Un Grove PIR Motion Sensor
    - Une Grove Chainnable RGB LED.

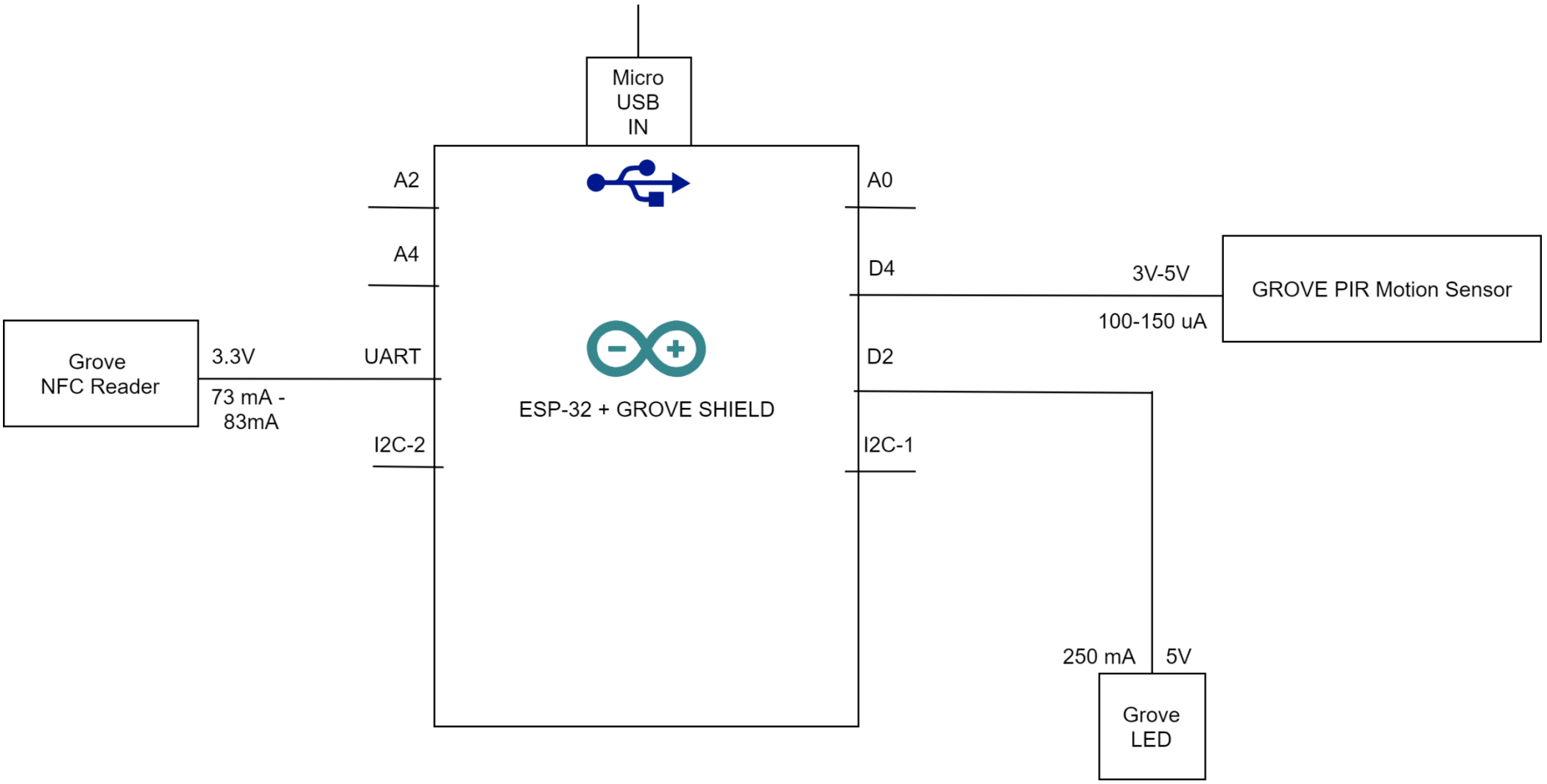


Figure 9 : Schéma de branchement

4.2. Librairies utilisées

Nous avons utilisé quelques librairies :

- ChainableLED.h, qui permet de gérer la LED Grove,
- WiFi.h, qui permet les opérations WiFi : connexion à un réseau par exemple,
- ArduinoJson.h, qui permet la sérialisation et désérialisation d'objets en JSON.
- HTTPClient.h, qui permet de lancer un serveur HTTP qui va écouter sur un port donné.
- NfcAdapter.h, qui permet la gestion du Grove NFC Reader.

4.3. Gestion des interruptions et *Tasks*

Pour ce projet, il nous a fallu gérer les interruptions matérielles au niveau du PIR Motion Sensor. En effet, ce dernier provoque une interruption matérielle dès lors qu'il détecte une présence.

Également, nous nous sommes familiarisés avec l'OS de l'ESP32 pour notamment mettre en place des *Tasks* (aussi appelés *Threads* plus couramment). Cela permettait de gérer en arrière-plan différentes actions. Pour plus de détails, consulter le code source sur github.

4.4. Déploiement de l'applicatif

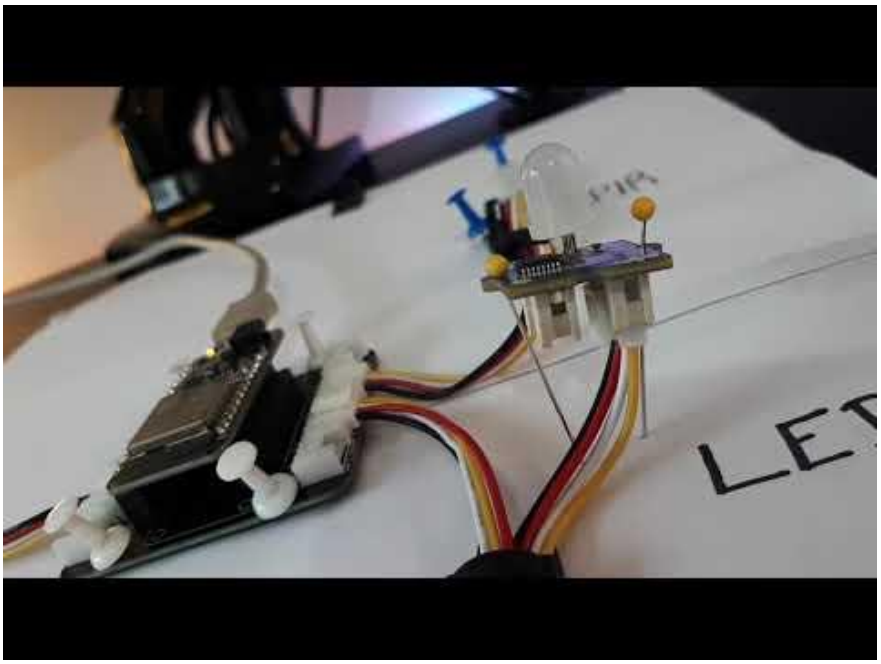
4.4.1. Le serveur

Le serveur tourne sur un PC et dispose d'une IP fixe, lui permettant d'être appelé par le serveur et le micro contrôleur.

4.4.2. L'application mobile

L'application mobile peut être utilisée sur téléphone grâce à l'apk générée, ou bien sur ordinateur à l'aide d'un émulateur.

5. PHASE 5 : DÉMONSTRATION FINALE



CONCLUSION

Ce projet fut riche en apprentissage. Il a été intéressant de pouvoir interconnecter une multitude de systèmes très hétérogènes, via le protocole HTTP. Ce projet nous a demandé de faire preuve de rigueur, car une simple erreur sur l'un des composants du système peut avoir des répercussions sur l'ensemble de l'écosystème applicatif.

Nous n'avions jamais travaillé avec des composants Arduino et sur un projet en C++, la partie systèmes embarqués était alors très formatrice pour nous.

Également, nous avons pu découvrir un nouveau framework de serveur HTTP JavaScript (NodeExpress), et le système de gestion de base de données MongoDB.

Finalement, nous avons pu confirmer nos acquis en matière de programmation mobile Android en Kotlin.