

# RAPPORT DE PROJET

---

APPLICATION MOBILE ANDROID

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

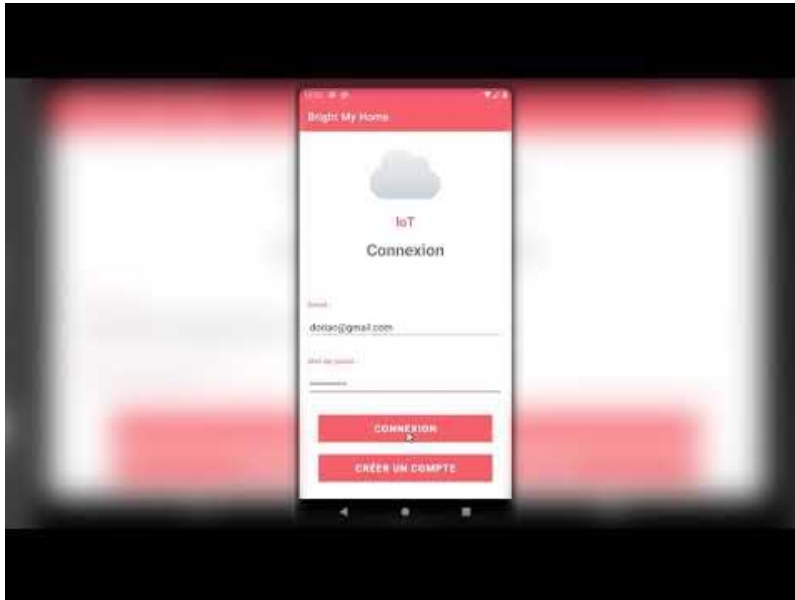
# INTRODUCTION

Dans le cadre de notre projet d'Application Mobile et d'IoT & Systèmes Embarqués, nous avons eu pour objectif de produire une application mobile Android avec le langage Kotlin et l'IDE Android Studio.

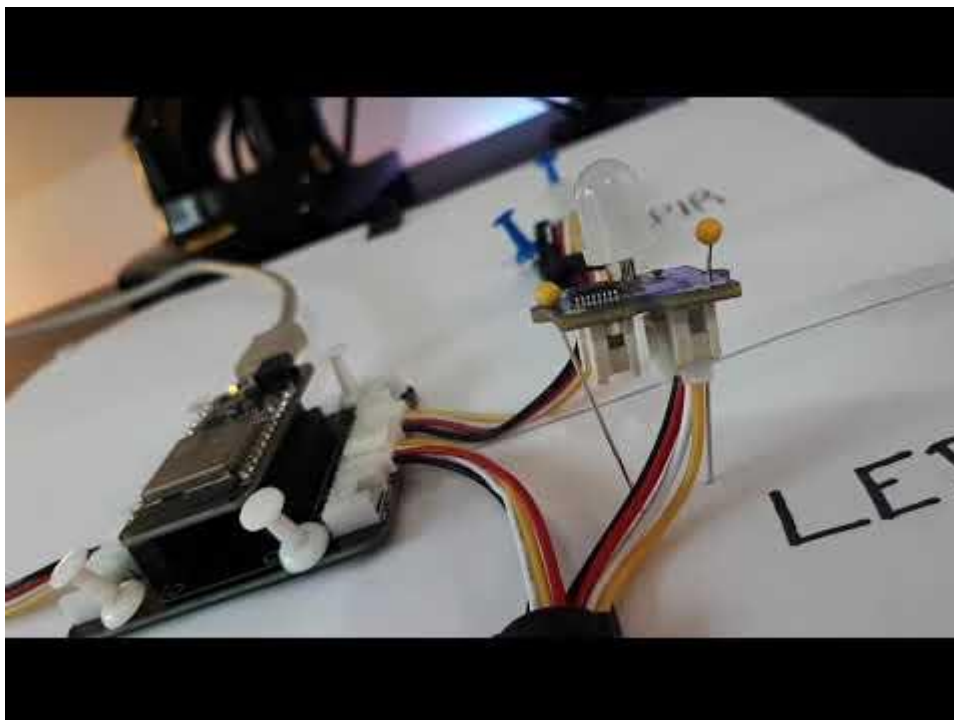
Notre application mobile permet la gestion et la création de compte, et la connexion à un dispositif d'éclairage automatique.

## 1. VIDÉOS DE PRÉSENTATION

Nous avons réalisé une vidéo qui présente les différents parcours utilisateurs possibles pour notre application dans sa globalité.



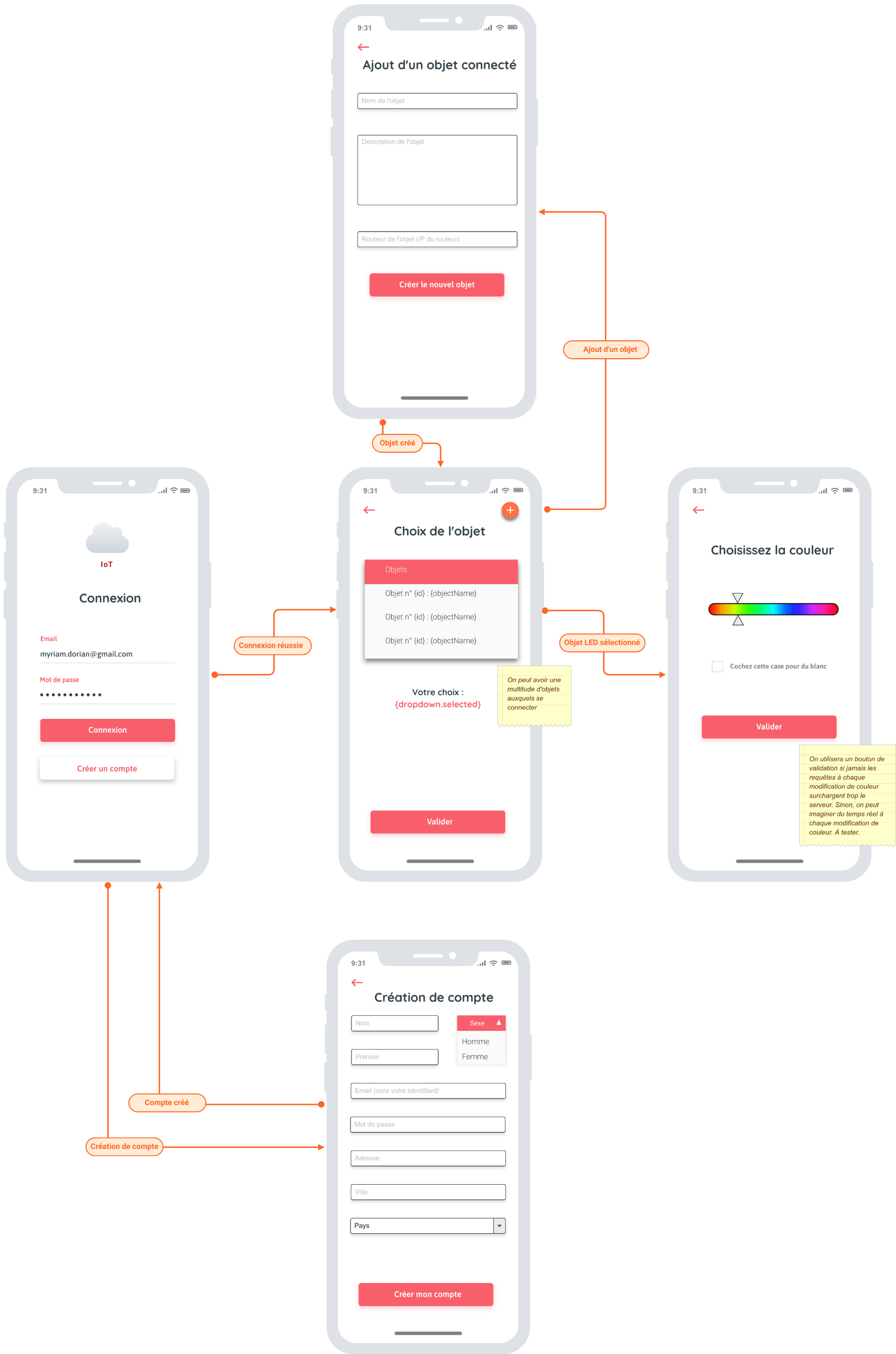
Une seconde vidéo, plutôt axée sur la partie IoT, n'est pas le sujet principal ici mais voici son lien ci-dessous. Des timestamps au sein de la vidéo donnent les différentes parties ; il est possible d'aller directement à la démonstration.



## 2. L'APPLICATION ANDROID

### 2.1. Conception

Pour notre application mobile, nous avons décidé dans un premier temps de concevoir l'ensemble des écrans dans un *wireframe* (maquettes).



## 2.2. Connexion & création de compte

L'utilisateur a la possibilité, lorsqu'il arrive sur notre application, de se connecter ou de créer un compte.

Pour se connecter, l'utilisateur doit entrer son email et son mot de passe s'il dispose déjà d'un compte. Une vérification est ensuite faite avec notre serveur Node.js (NodeExpress) pour confirmer que les identifiants sont corrects.

Sinon, l'utilisateur a la possibilité de créer un compte. Pour ce faire, il doit renseigner :

- Nom,
- Prénom,
- Sexe (sous forme de **Spinner**),
- Date de naissance grâce à un **Date Picker** et nous calculons l'âge de l'utilisateur que nous affichons à l'écran,
- Email,
- Mot de passe,
- Adresse,
- Ville,
- Pays (sous forme de **Spinner**).

Il peut ensuite s'enregistrer et utiliser son compte pour se connecter

## 2.3. Liste des objets connectés

Nous affichons dans notre vue principale la liste des objets connectés dans un **RecyclerView**. L'utilisateur peut ensuite cliquer sur un des objets pour accéder à son interface correspondante. Pour l'instant, chaque objet listé renverra sur la même et unique interface, celle de gestion de l'objet connecté produit dans le module d'IoT (gestion d'éclairage). Mais il est totalement possible par la suite de changer cela ; l'application est donc générique et modulaire.

## 2.4. Ajout d'objet connecté

Notre interface permet également l'ajout d'un nouvel objet connecté, en entrant notamment les informations suivantes :

- Nom de l'objet,
- Description de l'objet,
- IP du routeur WiFi auquel est connecté l'objet.

## 2.5. Interactions avec un objet connecté

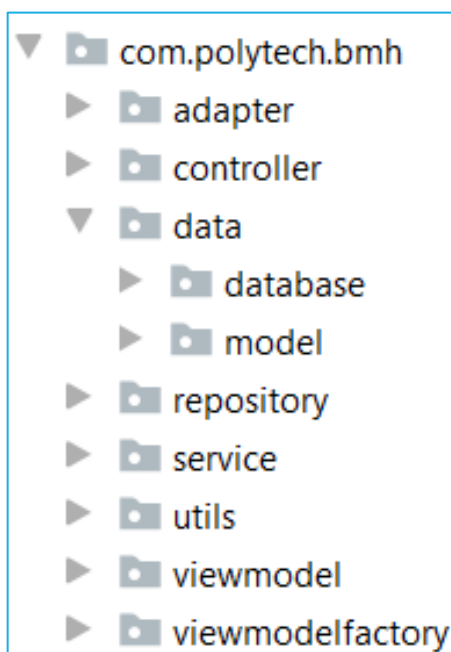
Notre interface permet d'interagir avec un objet connecté lorsqu'il a été sélectionné dans le **RecyclerView**. Lors du click sur un élément du RecyclerView, l'interface nous affiche le nom de l'objet, l'IP du routeur WiFi auquel il est connecté et la couleur de la led actuelle.

La seule interaction définie désormais est la gestion d'éclairage d'un système ESP32. L'interface dispose alors d'un **color picker** qui permet de changer la couleur de l'éclairage auquel est relié notre microcontrôleur ESP32.

### 3. ARCHITECTURE DE L'APPLICATION

Côté Kotlin, le projet est composé de huit packages. En effet, nous avons voulu séparer les préoccupations en différents packages.

Les packages sont :



- **controller** : il contient notre activité principale et tous nos fragments, ils permettent de gérer l'interface graphique d'application.
- **viewmodel** : il contient toutes les classes de type *ViewModel*, elles vont permettre de fournir à la vue associée les données utilisées par l'interface graphique. Dans le but d'observer les changements des données en respectant le cycle de vie de du contrôleur (activités et fragments), ces données sont de type *LiveData*.
- **viewmodelfactory** : il contient toutes les classes de type *ViewModelProvider.Factory*, elles vont permettre de transmettre des données au *ViewModel*.
- **repository** : il contient toutes les classes permettant de faire le lien entre le *ViewModel* et les sources de données).
- **service** : il contient l'instance de Retrofit et les interfaces permettant de communiquer avec notre API. Ce sont les fournisseurs de données provenant de notre API.
- **utils** : il contient les classes utilitaires. Dans notre cas, nous avons uniquement une méthode permettant de cacher le clavier du smartphone lorsque nous faisons certaines opérations.
- **data** : il est lui-même composé de deux packages :
  - o **database** : il contient le DAO et l'objet d'instance de la base de données locale (Room).
  - o **model** : il contient les classes métier.
- **adapter** : il contient un adaptateur qui permet de faire le lien entre la vue et les données, il prépare les données et leur affichage. Il contient également un *listener* qui implémente la fonction *onClick()*. (Utilisé pour le *RecyclerView*).

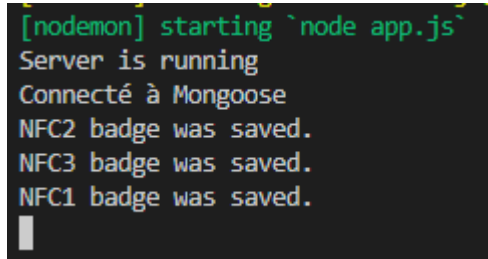
De plus, nous avons également les fichiers de ressources de l'application, notamment nos fichiers de code de l'interface graphique qui sont de type XML et qui sont présents dans le dossier `res/layout`.

## 4. INSTRUCTIONS D'EXÉCUTION

### 4.1. Configuration du serveur Node Express

Pour pouvoir lancer l'application et tester les fonctionnalités de base, il faut à minima :

- Installer un serveur MongoDB local (port 27017). [Windows : [Installation](#)]
- Cloner le serveur en local (git clone) : <https://github.com/IoT-PolytechLyon/iot-2a-server>
- Se placer dans le dossier du serveur et lancer la commande ``nodemon app.js``
- Si tout s'est bien passé, le serveur affiche :



```
[nodemon] starting `node app.js`  
Server is running  
Connecté à Mongoose  
NFC2 badge was saved.  
NFC3 badge was saved.  
NFC1 badge was saved.
```

- Le serveur est prêt et tourne sur le port 8081.

### 4.2. Lancement de l'application mobile

Sous Android Studio, il sera ensuite possible de tester l'ensemble de l'application une fois que le serveur tourne. Sinon, seules les premières pages de connexion et de création de compte seront disponibles. La configuration Android qui pointe vers le serveur Node Express se situe dans le fichier « `RetrofitInstance.kt` » du package « `service` ».

## CONCLUSION

Ce projet nous a permis d'en apprendre beaucoup sur l'écosystème Kotlin + Android, et sur l'architecture préconisée par Google lors du développement d'application mobiles. Nous avons notamment utilisé des fragments, *viewmodels*, *repositories*, *services*...

Également, côté IoT, nous avons pu apprendre à interconnecter de nombreux systèmes ensemble et ce fut fort intéressant.

Cependant, Android Studio est un logiciel très lourd à utiliser, rendant le travail difficile sur des petites configurations. Également, le langage Kotlin propose beaucoup de nouveautés dans la syntaxe, ce qui peut être difficile pour s'habituer lorsque l'on passe de l'un à l'autre, notamment côté framework et appels de fonctions spécifiques (*streams* entre autres).

Le langage se veut également moins « verbeux » que Java, et donc plus concis.