



Rapport du projet de groupe HODOR

Raphaël GUIOT – Wilfrid MEZARD | LP IOTIA | 11 Mai 2020

Table des matières

[**Présentation du projet** 2](#_Toc40016644)

[**Cahier des Charges** 3](#_Toc40016645)

[Fonctionnalités 3](#_Toc40016646)

[architecture materielle 4](#_Toc40016647)

[architecture logicielle 5](#_Toc40016648)

[Mécanisme de chiffrement employé 6](#_Toc40016649)

[scenarii 7](#_Toc40016650)

[**Codes réalisés et explications** 7](#_Toc40016651)

[Arduino 7](#_Toc40016652)

[PHP 10](#_Toc40016653)

[python 10](#_Toc40016654)

[**Bilan** 10](#_Toc40016655)

# **Présentation du projet**

Notre projet étudiants, baptisé projet HODOR (en référence au personnage de la série américaine *Game Of Thrones* et de la scène mythique de l’épisode 5 de la saison 6 - https://youtu.be/6BNDJd7S0Pc?t=142) consiste en un système d’ouverture et de fermeture d’une porte sans clé et à distance.

Notre souhait est de mettre à disposition une application basée sur un lecteur RFID utilisant protocoles de communication en sécurisant autant que faire se peut les échanges entre les différents éléments de notre système.

Dans un premier temps, notre réflexion s’est axée sur la mise en place d’un lecteur RFID, d’un broker MQTT et d’un serveur d’application Web. Nous avons donc estimé le matériel nécessaire à la mise en place de notre projet :

* Carte ESP8266
* Lecteur RFID et jeu de cartes/badges
* Raspberry Pi
* Arduino Nano BLE 33
* Servo moteur
* Jeu de LED
* Buzzer

Le cadre de notre projet ayant pour contrainte de s’adapter aux trois cours proposés par *Marie Agnès PERALDI*, *Cécile BELLEUDY* et *Jean-Michel BRUNEAU* nous avons dû respecter certaines contraintes parmi lesquelles la mise en place d’une application Web développée en PHP, de l’utilisation de protocoles d’échanges tels que le Bluetooth Low Energy ou encore la sécurisation des échanges entre nos différents composants.

L’autre point qui a grandement influencé notre organisation de travail concerne le confinement auquel nous sommes soumis dans un contexte de pandémie mondiale de COVID-19. Dans l’impossibilité de se retrouver pour mettre en place notre projet, nous avons dû nous adapter afin de limiter au maximum l’impact sur notre travail et de proposer aux professeurs un rendu en adéquation avec leurs attentes.

C’est pourquoi nous avons choisis dans le présent rapport de détailler autant que cela est possible notre démarche et les problématiques abordées lors du développement de notre projet.

# **Cahier des Charges**

Comme cela a pu être détaillé dans notre cahier des charges, nous reprenons ici les éléments essentiels qui nous ont servi à orienter le développement de ce projet étudiant.

## Fonctionnalités

Dans un premier temps, nous avons découpé le projet en fonctionnalités. Ces fonctionnalités ont été modifiées et réfléchies tout au long du développement est peuvent être résumées tel que dans la liste suivante :

* F1 – ouverture via lecteur RFID
* F2 – ouverture via connexion sur le serveur Web (HTTPS)
* F3 - définition de plages horaires
* F4 – entretien de log d’activité sur la porte (ouverture, fermeture, changement d’autorisation)

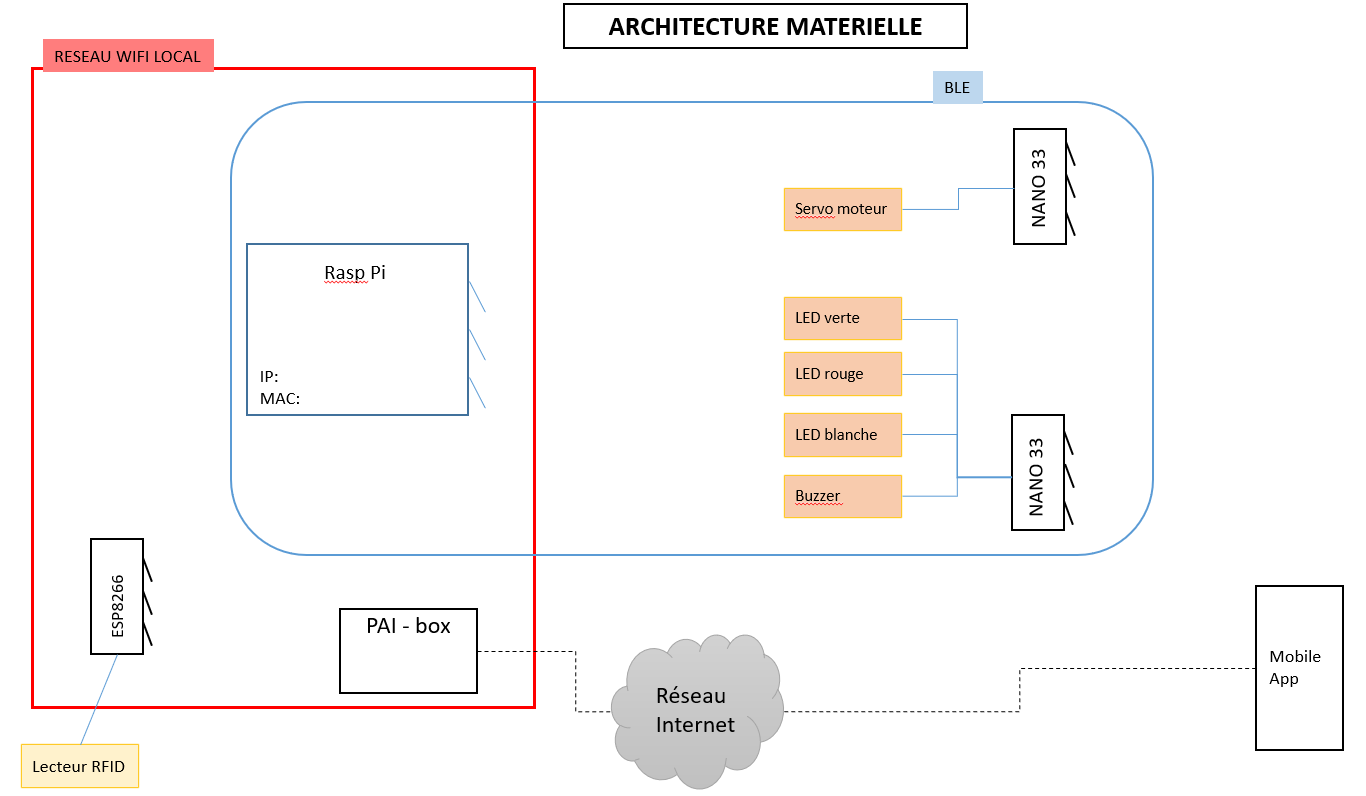
Fonctionnalités souhaitées initialement et qu’il reste à développer

* F5 – ouverture via adresse MAC (d’un mobile par exemple). Non réalisé car énorme souci de sécurité

En considérant notre effectif restreint (deux personnes) et nos connaissances préalables (aucunes connaissances du langage Python, découverte récente de la programmation pour Arduino, découverte de PHP pour Wilfrid) il nous a paru essentiel de se concentrer principalement sur certains aspects du développement et de négliger certaines parties qui, bien qu’essentielles, pourront être compléter par la suite sans que cela ne porte atteinte aux développements déjà effectués.

Il nous est paru indispensable, dans le contexte particulier, de nous concentrer sur certains aspects du projet et d’adapter notre projet au travail en équipe à distance.

## architecture materielle



Tel que cela a pu être détaillé lors des échanges réguliers avec nos professeurs, nous souhaitons organiser notre projet tel que présenté dans le présent schéma.

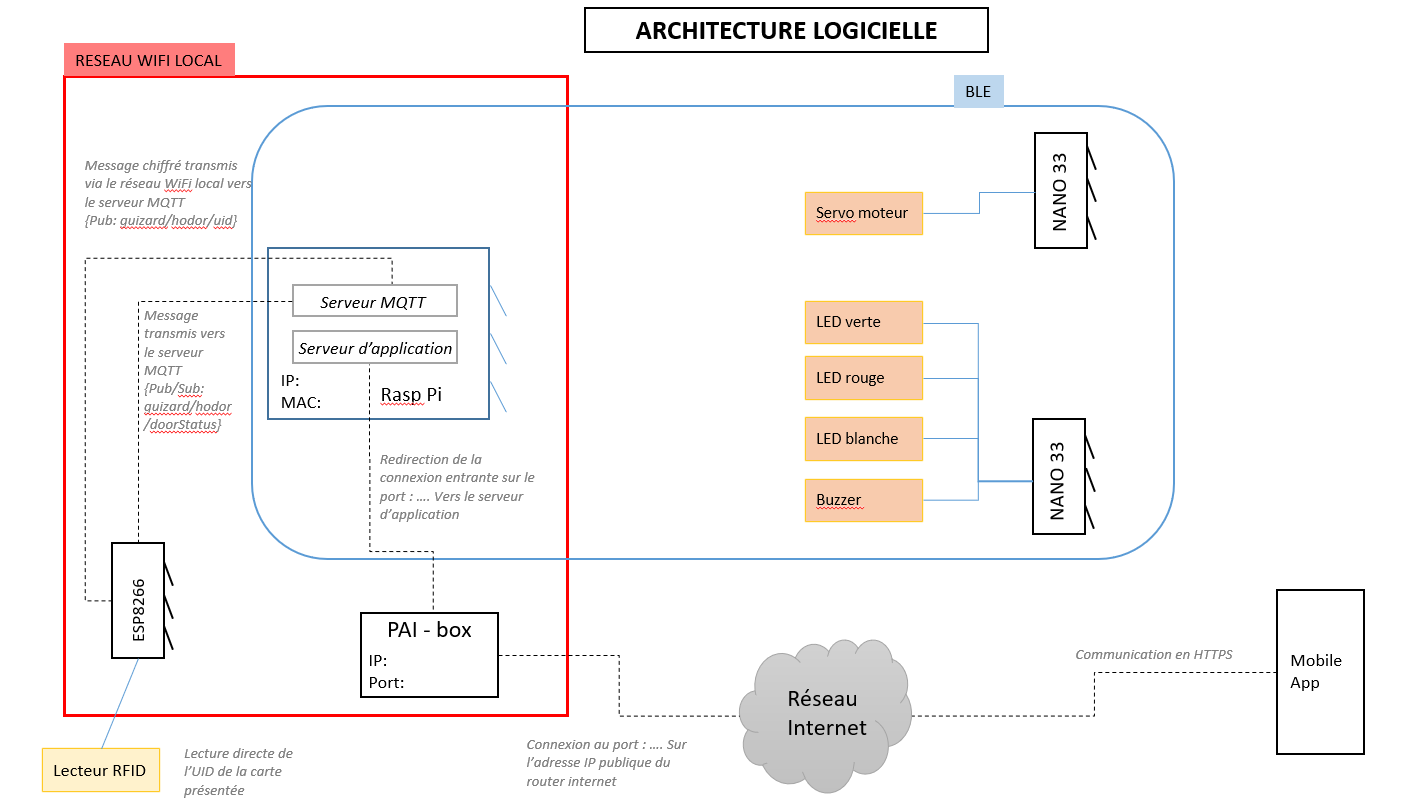
Notre souhait étant de multiplier les différents protocoles de communication au sein d’un même projet, nous avons mis en œuvre un réseau local WiFi permettant de mettre en communication notre ESP (carte sur laquelle est positionné le lecteur RFID) avec le Raspberry Pi (qui joue le rôle central de notre projet) mais également de rendre disponible le serveur d’application Web au réseau internet.

Pour ce qui concerne les actionneurs, nous avons décidé de nous appuyer sur des échanges basés sur le Bluetooth Low Energy et sur l’utilisation de deux cartes Arduino Nano BLE 33. Aux vues de nos capacités et de nos connaissances sur ce point particulier, nous avons considéré que, bien qu’essentiel, l’avancement du projet sur cette partie reste limité et n’est pas encore fonctionnel.

Il est nécessaire de souligner que notre application Web n’a pas été développée dans l’optique de déployer une application mobile à proprement parler mais comme une application Web responsive accessible aussi bien via le navigateur d’un ordinateur comme celui d’un smartphone.

Les échanges et leurs détails seront vus par la suite dans la partie concernant l’architecture logicielle du projet).

## architecture logicielle



Comme cela a pu être expliqué précédemment, notre souhait initial était de multiplier les protocoles de communication utilisés dans notre projet. Le schéma présenté ici permet de mettre en lumière les échanges et les protocoles utilisés.

Notre attention s’est portée dans un premier temps sur l’utilisation d’un broker MQTT et sur un réseau WiFi local mettant en communication notre ESP avec Raspberry Pi. Nous nous sommes rapidement aperçut que du point de vue de la sécurité, l’utilisation d’un broker laisse à désirer. En effet la sécurité de l’échange se base sur l’utilisation d’un jeu d’**users** et de **passwords** pour limiter l’accès à un ou des topics MQTT. Quiconque ayant accès à ces informations peut alors souscrire et publier au topic sur lequel sont échangés les clés UID lue en RFID d’où notre volonté de « chiffrer » l’échange et par la même occasion d’approfondir nos connaissances en ce qui concerne la manipulation des types de données dans la programmation bas niveau des Arduino.

### Mécanisme de chiffrement employé

Le mécanisme de chiffrement est somme toute basique et peut être résumé tel que :

Soit une clé UID lue par le lecteur RFID. Par définition cette clé est une suite de 8 caractères (par exemple : 879616A3).

La première étape de notre mécanisme de « chiffrement » consiste en la permutation deux à deux des 8 caractères :



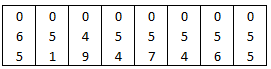
Devient alors :



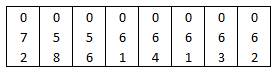
Puis l’on inverse l’ordre des caractères :



Les caractères étant toujours clairement identifiables, il convient d’agir sur le codage ASCII des caractères en le modifiant à l’aide d’une « clé secrète » partagée par le Raspberry Pi et l’ESP. Dans notre projet nous l’avons fixé à key = 7, mais il convient d’envisager de modifier cette clé et de l’envoyer en même temps que notre message pour plus de sécurité. Dans notre exemple cela donne :



Transformé en :



Qui donne finalement la clé transmise suivante :



## scenarii

* Propriétaire/invité utilise F1
* Propriétaire/invité utilise F2
* Propriétaire utilise F3

Non réalisé mais souhaité :

* Propriétaire/invité utilise F5

# **Codes réalisés et explications**

## Arduino

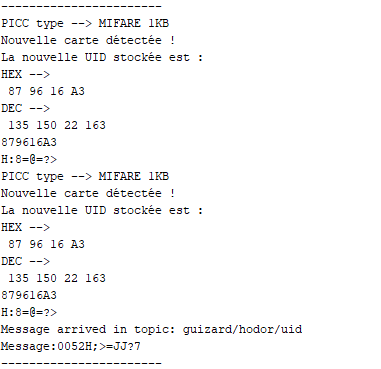
L’ensemble des codes Arduino se trouvent sous le dossier ‘ARD’ sur dépôt git du projet, ils ont été réalisés par Wilfrid Mezard.

***Fichier LECT\_RFID.ino :***

Connexion au broker MQTT, lecture RFID, chiffrage et envoi de l’UID lue vers le Raspberry Pi via le broker MQTT (topic : « guizard/hodor/uid »).

Ce code est placé sur l’ESP8266 et permet la lecture de l’UID d’une carte/badge et son envoi chiffré vers le Raspberry Pi.





Dans un soucis de sécurité et comme cela é déjà été précisé, la clé UID n’est pas transmise en clair via le broker MQTT.

De plus, nous avons choisi d’insérer quatre caractères variables dans la trame transmise. Dans une première étape du développement, nous souhaitions insérer une date récupérée via un web service et l’intégrer dans la trame (Le code de cet essai se trouve sous le dossier ‘TEST’ – test\_WiFi\_co\_mqtt\_send\_ui\_card\_HOME.ino - de notre dépôt git). Malheureusement, les incertitudes sur la fiabilité du signal WiFi et les erreurs ponctuelles retournées par le web service nous ont fait abandonner cette idée.

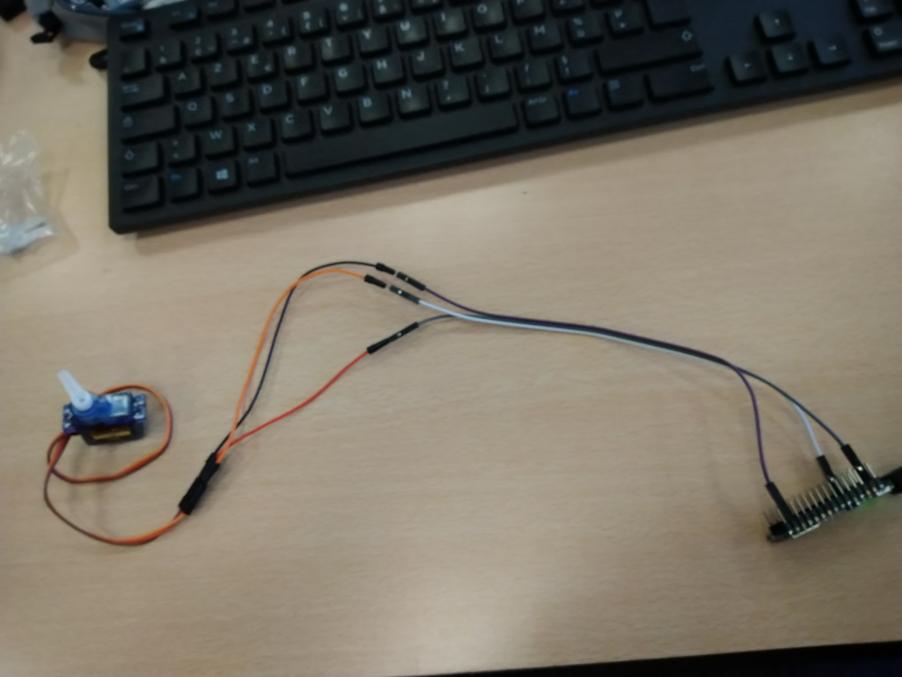
En revanche souhaitant garder une portion variable dans notre trame transmise, nous avons choisis d’intégrer 4 caractères numériques modifiés à chaque loop() de notre Arduino, ce qui donne le genre de trame suivante :



Bibliothèques utilisées et dépendances à récupérer :

* ESP8266WiFi.h : Permet l’utilisation de la WiFi
* PubSubClient.h : Permet la souscription et la publication via un broker MQTT
* SPI.h et MFRC522.h : Permettent la lecture d’une carte/badge RFID pour l’ESP

***Fichier SERV.ino :***

Ce code est positionné sur l’une des cartes Arduino Nano BLE 33 et permet l’utilisation d’un servo moteur simulant notre loquet de fermeture de la porte. Tel quel, l’ouverture/fermeture de la porte se fait via la lecture du port série ce qui n’est pas notre souhait final. En effet, l’actionnement de notre système est basé sur une communication Bluetooth Low Energy et les commandes/actions à effectuer sont renseignées via ce protocole.

Montage classique d'un servo moteur sur Arduino Nano BLE

Bibliothèque utilisée :

* Servo.h : Permet l’utilisation d’un servo moteur via une instance de Servo

Ce code n’est donc pas, en l’état, utilisé ici et doit encore être « fusionné » avec l’utilisation de la communication BLE telle que décrite par la suite. Un autre point n’est pas traité ici et concerne la fermeture de la porte après un certain laps de temps après une ouverture (point fondamental pour la suite pour ce qui concerne l’état réel du statut de la porte).

***Fichier doorStatusBLE.ino :***

Ce fichier doit être combiné avec le précédent. En effet dans ce code-ci est mis en place la communication BLE d’un Arduino Nano BLE 33. Une des particularités du protocole BLE (hormis l’avantage qu’il présente sur le plan énergétique) est la vue « protocolaire » qui est faite de son usage et plus particulièrement le protocole GATT.

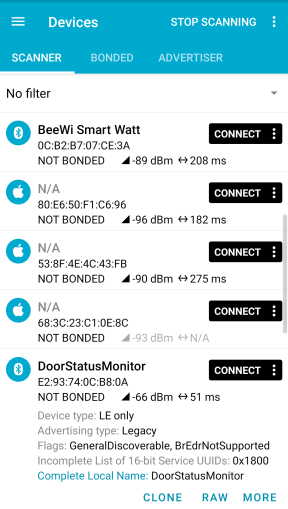
L’idée simplifiée est d’établir une relation maître-esclave entre un serveur GATT (l’esclave) et un client (le maître), le serveur exposant à un client des données (des *attributes* présentés soit sous la forme de *services* soit sous la forme de *characteristics*).

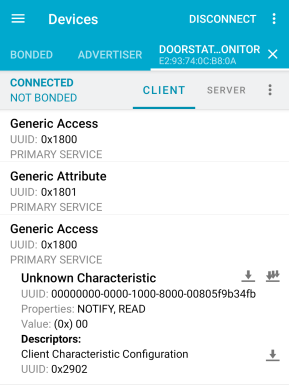
Dans le code présenté par exemple on souhaite renseigner, à la demande, l’état de la porte (ouverte ou fermée).

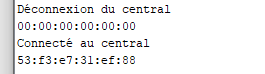
Ce besoin est issu de l’utilisation de l’application Web développée par Raphaël. Après discussion il nous est apparu que connaître le statut de la porte à l’instant *t* est nécessaire mais nous ne souhaitons pas stocker en base de données cette notion. En effet dans le cas d’un évènement fortuit ou non prévu, il est probable qu’une désynchronisation de l’état de la porte survienne entre l’état réel et l’état stocké en base de données.

Par conséquent, on a souhaité recourir à un script Python (détaillé par la suite) pour utiliser une communication BLE et récupérer l’état réel de la porte.

Ainsi dans notre cas, il a fallu définir un **service** (*Generic Access*) qui présente une **characteristic** en lecture/notification et présente donc l’état réel de la porte.







Lecture du port série lorsqu'une connexion BLE est établie

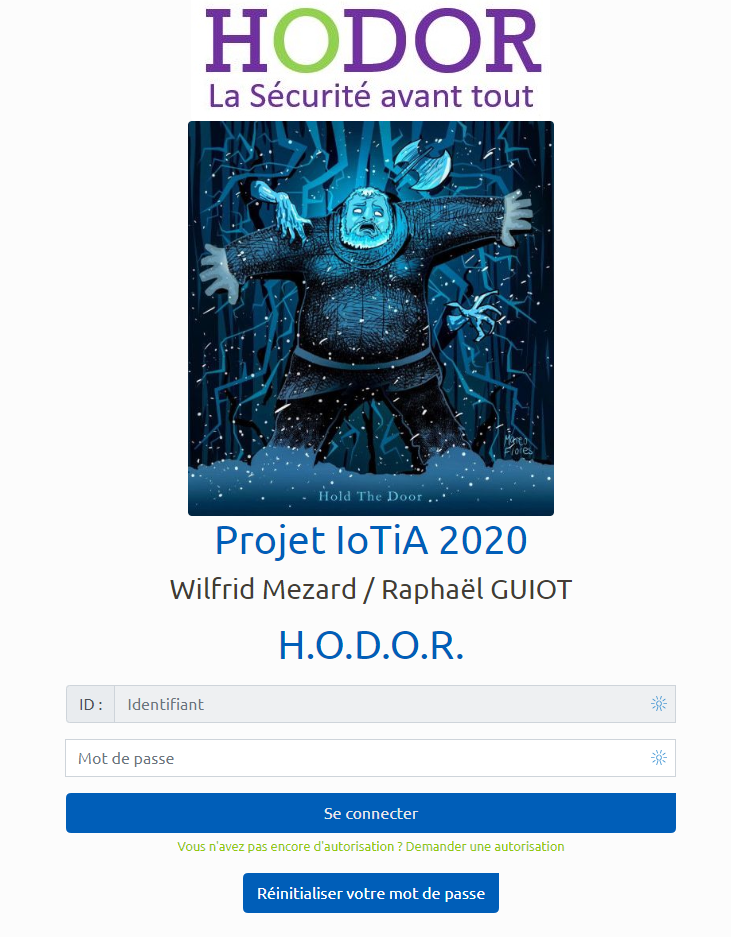
Bibliothèque utilisée :

* ArduinoBLE.h : Cette librairie permet de mettre en place une communication BLE pour un composant Arduino. Elle permet en plus de respecter des spécifications propres au BLE (voir le site internet - [www.bluetooth.com/specifications/gatt/](http://www.bluetooth.com/specifications/gatt/) pour ce qui concerne les *Bluetooth Generic Attributes* GATT)

## PHP

L’ensemble des codes relatifs au serveur Web et à l’application se trouvent sous le dossier ‘AppMobile’ du dépôt git du projet, ils ont été réalisés par Raphaël Guiot.

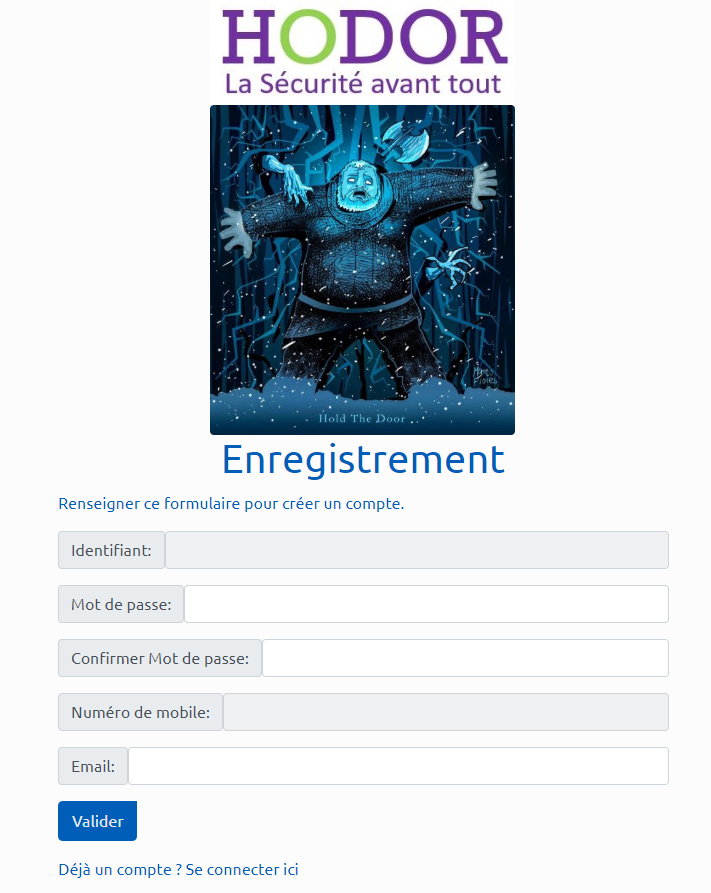
***Accueil sur l’AppMobile (AppMobile/login.php) :***



L’application est disponible à cette adresse : <https://90.116.66.46:8443> , l’accès en http ayant désactivé et seul l’accès en HTTPS est autorisé.

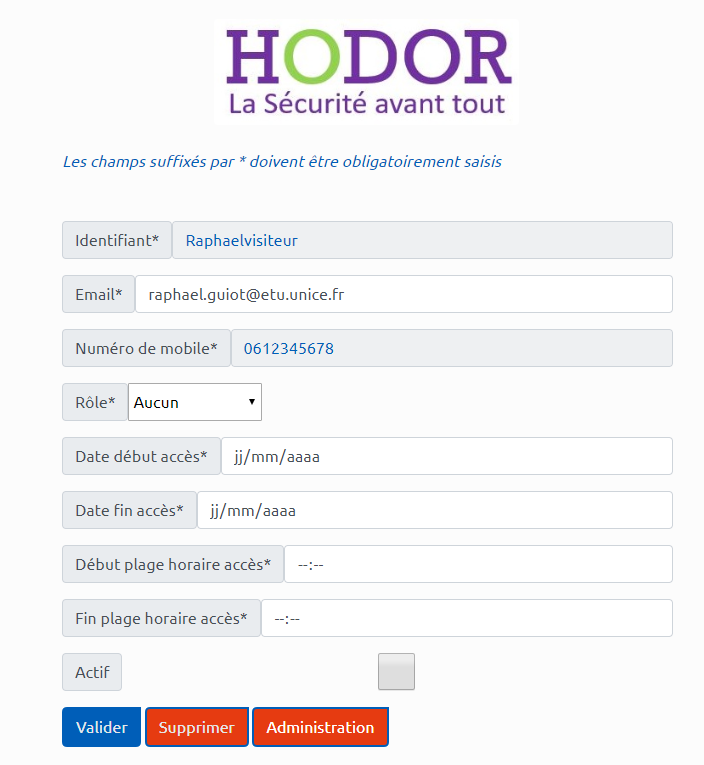
***Création et validation d’un compte sur l’AppMobile :***

Dans un 1er temps, le visiteur a besoin de remplir un formulaire pour s’enregistrer en base de données :



Une fois le formulaire envoyé et la demande d’autorisation envoyée, un administrateur devra par l’intermédiaire de la page Administration valider son compte lui donner un rôle et définir les dates et plages horaires autorisées pour le visiteur :





***Accès panneau de contrôle de l’AppMobile :***

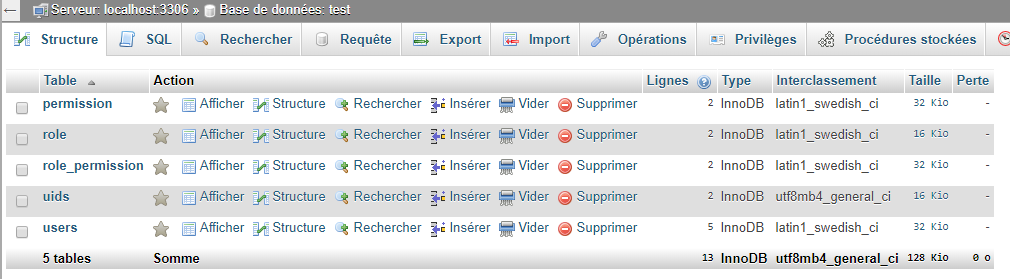
Une fois le compte validée par un administrateur le visiteur aura accès au panneau de contrôle qui lui permettra d’actionner l’ouverture de la porte ou sa fermeture :



L’action sur l’input switch affichera un message de confirmation javascript qui décrira l’action qui a été envoyé au broker MQTT pour déclencher la séquence d’ouverture/fermeture aux arduinos.

***Fonctionnement de l’AppMobile et techno utilisées :***

L’application a été entièrement codé en php (version 7.3) et utilise la librairies bootstrap 4.4.1 pour la partie front et le côté responsive pour les différents devices qui y accèderont. Pour l’utilisation de la fonction de broker MQTT côté php il a été nécessaire de modifier le noyau de php pour activer le module Mosquitto et pouvoir utiliser sa classe associée et permettre à l’application côté php de pouvoir publish/subscribe à nos topics.

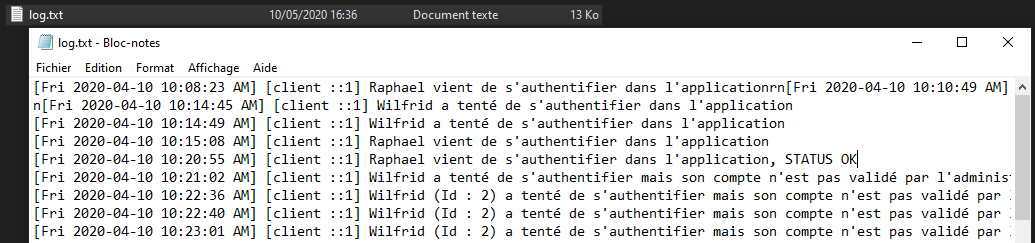
Derrière nous avons une base de données MariaDB 10.3.22 – Raspbian 10

Dans laquelle nous stockons la liste des utilisateurs, les cartes RFID (‘uids’) validées et les rôles/permissions.

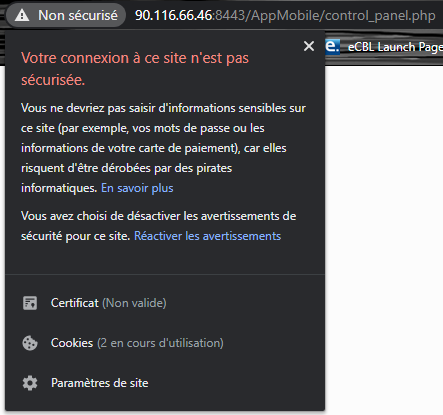
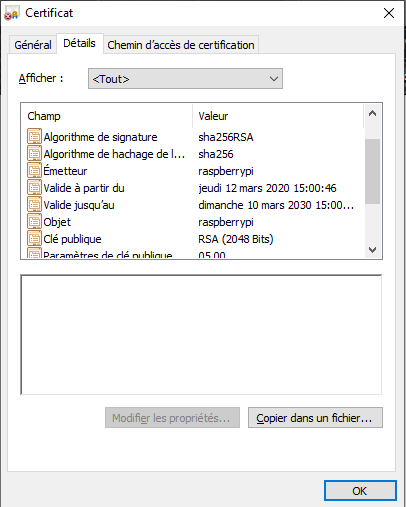
Côté php nous utilisons la méthode PDO grâce au fichier de config\_db.php qui contient les paramètres d’authentification à la base et crée un objet PDO qui nous servira pour accéder aux informations de la BDD.

Dans la partie login.php nous faisons de multiples contrôle sur l’username, le password envoyé par le formulaire puis nous vérifions à l’aide d’une succession de conditions que le statut du compte est validé (égal à 1) puis que la date courante (correspondante au moment de l’authentification) est comprise entre les 2 dates définies par l’administrateur pour ce visiteur et enfin que l’heure de connexion est comprise entre les plages horaires définies par celui-ci encore.

De plus des contrôles sont fait sur le contenu des champs + ajout de la fonction htmlspecialchars() pour éviter que ces champs contiennent des caractères interprétables de même que pour la partie register.php et admin\_user.php

De plus une fonction my\_log est définie et enregistre dans un fichier texte /my\_log/log.txt les activités sur l’application.

De plus l’accès en HTTPS est activé et son accès en HTTP désactivé, le certificat est bien crée mais comme non validé par une autorité de certification, le navigateur annonce un problème de sécurité lors de la 1ère connexion :

## python

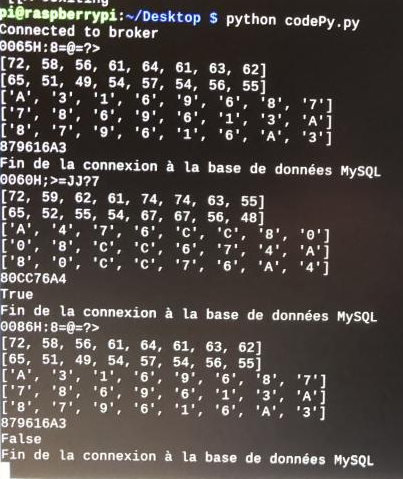
L’ensemble des codes Python utilisés se trouvent sous le dosser ‘PYTH’ du dépôt git du projet, ils ont été réalisés par Wilfrid Mezard.

Au cours du développement de notre projet une problématique est rapidement apparue sur les interfaces à mettre en œuvre pour permettre la communication entre nos différents composant et par la même occasion la nécessité d’utiliser des scripts écrits en python pour répondre à ce point. Etant tout deux issus d’une reconversion professionnelle, nos connaissances en Python sont très limitées et ce point a été pour nous l’occasion de renforcer nos compétences sur ce langage.

***Fichier codePy.py :***

Ce script permet la mise en place d’une communication entre l’ESP et le Raspberry Pi. En effet connectés via le réseau WiFi local (dans notre situation et dans le cadre du confinement, nous avons été obligés de travailler à distance en utilisant une IP publique et une redirection de port) la clé chiffrée est transmis via le protocole MQTT et il convient de mettre en place cette communication via un script python.

Dans la version finalisée de notre projet, ce script est appelé lors du boot du Raspberry Pi et doit tourner en permanence, à l’écoute d’une publication sur le topic dédié par l’ESP. Concrètement, le script met en place une connexion au broker MQTT (client « Python ») et souscrit au topic « guizard/hodor/uid » dans l’attente d’une clé chiffrée. Lorsqu’une clé est scannée, la clé chiffrée est publiée et, via la clé partagée entre le Raspberry Pi et l’ESP, la clé est déchiffrée et une connexion à la base de données est mise en place pour aller vérifier la validité ou non de la carte scannée.



Dans cet exemple nous scannons donc deux clés et nous affichons dans la console leur autorisation d’accès (booléen).

Ce script Python reste néanmoins incomplet dans la mesure où si la carte possède un accès valide, une commande doit être transmise à l’actionneur via le protocole Bluetooth LE.

# **Bilan**

Bilan de ce qui a effectivement été fait.

Bilan des compétences qu’on a pu développer sur ce projet.

Mea Culpa sur ce qu’il reste à faire.