

**CAHIER DE BORD**

David Pigeon

ÉQUIPE

David Tremblay

Jean-Paul Bai

Florian Hostashy

PROJETS TECHNOLOGIQUES

420-009-RL

(H2023)

Projet DOMOTIQUE

Préparer, mettre en œuvre et évaluer

Une image contenant texte, équipement électronique, circuit

Description générée automatiquementUne image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Une image contenant équipement électronique, circuit

Description générée automatiquement





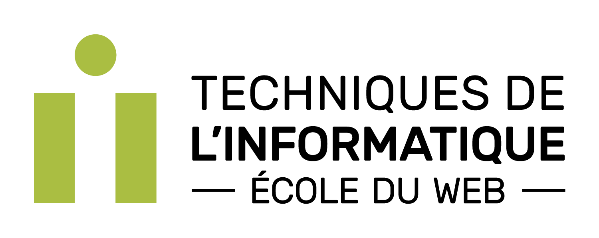
Une image contenant texte

Description générée automatiquementUne image contenant texte, clipart

Description générée automatiquementUne image contenant texte

Description générée automatiquementUne image contenant équipement électronique

Description générée automatiquement



# Table des matières

[Table des matières 1](#_Toc124502792)

[Équipe 5](#_Toc124502793)

[Identité 5](#_Toc124502794)

[Membres 5](#_Toc124502795)

[Demande 5](#_Toc124502796)

[Client 5](#_Toc124502797)

[Demande 6](#_Toc124502798)

[Conception 7](#_Toc124502799)

[Contexte 7](#_Toc124502800)

[Complément d’information 8](#_Toc124502801)

[Objectifs 9](#_Toc124502802)

[Objectif 1 : Programmer le Broker - Raspberry Pi 10](#_Toc124502803)

[Objectif 2: Construire et programmer des senseurs ESP32-MQTT-Wifi 11](#_Toc124502804)

[Objectif 3: Construire et programmer des actualisateurs ESP32-MQTT-Wifi 13](#_Toc124502805)

[Objectif 4 : Installer et configurer un **senseur** MQTT-ZigBee 15](#_Toc124502806)

[Objectif 5 : Installer et configurer un **actualisateur** MQTT- ZigBee -RELAY 16](#_Toc124502807)

[Objectif 6 : Programmation native 17](#_Toc124502808)

[Veille technologique 25](#_Toc124502809)

[Raspberry Pi (voir annexe 1) 25](#_Toc124502810)

[Mqtt (voir annexe 2) 25](#_Toc124502811)

[ZigBee2MQTT (voir annexe 3) 25](#_Toc124502812)

[Senseur RFiD-ESP32 (voir annexe 4) 25](#_Toc124502813)

[Senseur de présence-ESP32 (voir annexe 5) 25](#_Toc124502814)

[Senseur de luminosité-ESP32 (voir annexe 6) 25](#_Toc124502815)

[Senseur de CO2-ESP32 (voir annexe 7) 25](#_Toc124502816)

[Senseur de Gaz-ESP32 (voir annexe 8) 25](#_Toc124502817)

[Senseur de température-Zigbee (voir annexe 9) 25](#_Toc124502818)

[Senseur switch-Zigbee (voir annexe 10) 25](#_Toc124502819)

[Écran Stone (voir annexe 11) 25](#_Toc124502820)

[Contrôleur d’ouverture de porte Zigbee (voir annexe 12) 25](#_Toc124502821)

[Contrôleur de lumière Zigbee (voir annexe 13) 25](#_Toc124502822)

[Programmation d’Android native (voir annexe 14) 25](#_Toc124502823)

[Programmation iPhone native (voir annexe 15) 25](#_Toc124502824)

[Base de données (voir annexe 16) 25](#_Toc124502825)

[Stratégies de développement 26](#_Toc124502826)

[Répartition des tâches 26](#_Toc124502827)

[Organisation du travail 26](#_Toc124502828)

[Outils de partage (collaboration) 27](#_Toc124502829)

[Méthode de gestion de projet utilisée 27](#_Toc124502830)

[Gestion des versions 27](#_Toc124502831)

[Calendrier des livrables 28](#_Toc124502832)

[Mise en œuvre 29](#_Toc124502833)

[Références et documentation 29](#_Toc124502834)

[Liste des versions (Gits) 29](#_Toc124502835)

[Annexes – Veilles technologiques 30](#_Toc124502836)

[Annexe 1 –Raspberry Pi 30](#_Toc124502837)

[Caractéristiques 30](#_Toc124502838)

[Installation de l’OS 30](#_Toc124502839)

[Configuration et installation dans l’environnement de l’ÉDW 30](#_Toc124502840)

[Annexe 2 –Mqtt 30](#_Toc124502841)

[Technologie 30](#_Toc124502842)

[Installation d’un broker dans un Raspberry Pi 30](#_Toc124502843)

[Installation d’une fonctionnalité « subscriber » et « publisher » 30](#_Toc124502844)

[Dans un Raspberry Pi (Linux C++) 30](#_Toc124502845)

[Dans en ESP32 30](#_Toc124502846)

[Dans iPhone (Swift) 30](#_Toc124502847)

[Dans Android (Java) 30](#_Toc124502848)

[Annexe 3 –ZigBee2MQTT 30](#_Toc124502849)

[Technologie 30](#_Toc124502850)

[Installation dans un Raspberry Pi 30](#_Toc124502851)

[Utilisation d’un « Sniffer » Zigbee CC2531 30](#_Toc124502852)

[Annexe 4 –Senseur RFId-Esp32 31](#_Toc124502853)

[Références 31](#_Toc124502854)

[Schéma explicatif 31](#_Toc124502855)

[Schéma de branchement 31](#_Toc124502856)

[Programmations 31](#_Toc124502857)

[Libraires utilisées 31](#_Toc124502858)

[Connexion au Mqtt (« publisher ») 31](#_Toc124502859)

[Difficultés et solutions 31](#_Toc124502860)

[Documentation technique 31](#_Toc124502861)

[Bilan / conclusion 31](#_Toc124502862)

[Annexe 5 –Senseur de présence-Esp32 31](#_Toc124502863)

[Références 31](#_Toc124502864)

[Schémas 31](#_Toc124502865)

[Programmations 31](#_Toc124502866)

[Connexion au Mqtt (« publisher ») 31](#_Toc124502867)

[Annexe 6 –Senseur de luminosité-Esp32 32](#_Toc124502868)

[Références 32](#_Toc124502869)

[Schémas 32](#_Toc124502870)

[Programmations 32](#_Toc124502871)

[Connexion au Mqtt (« publisher ») 32](#_Toc124502872)

[Annexe 7 –Senseur de CO2-Esp32 32](#_Toc124502873)

[Références 32](#_Toc124502874)

[Schémas 32](#_Toc124502875)

[Programmations 32](#_Toc124502876)

[Connexion au Mqtt (« publisher ») 32](#_Toc124502877)

[Annexe 8 –Senseur de Gaz-Esp32 32](#_Toc124502878)

[Références 32](#_Toc124502879)

[Schémas 32](#_Toc124502880)

[Programmations 32](#_Toc124502881)

[Connexion au Mqtt (« publisher ») 32](#_Toc124502882)

[Annexe 9 –Senseur de température-Zigbee 33](#_Toc124502883)

[Références 33](#_Toc124502884)

[Schémas 33](#_Toc124502885)

[Programmations 33](#_Toc124502886)

[Connexion au ZigBee2Mqtt (« publisher ») 33](#_Toc124502887)

[Annexe 10 –Senseur Switch-Zigbee 33](#_Toc124502888)

[Références 33](#_Toc124502889)

[Schémas 33](#_Toc124502890)

[Programmations 33](#_Toc124502891)

[Connexion au ZigBee2Mqtt (« publisher ») 33](#_Toc124502892)

[Annexe 11 –Écran Stone 33](#_Toc124502893)

[Références 33](#_Toc124502894)

[Schémas 33](#_Toc124502895)

[Programmations 33](#_Toc124502896)

[Connexion au ZigBee2Mqtt (« subscriber ») 33](#_Toc124502897)

[Annexe 12 –Contrôleur de porte Zigbee 34](#_Toc124502898)

[Références 34](#_Toc124502899)

[Schémas 34](#_Toc124502900)

[Programmations 34](#_Toc124502901)

[Connexion au ZigBee2Mqtt (« subscriber ») 34](#_Toc124502902)

[Annexe 13 –Contrôleur de lumière Zigbee 34](#_Toc124502903)

[Références 34](#_Toc124502904)

[Schémas 34](#_Toc124502905)

[Programmations 34](#_Toc124502906)

[Connexion au ZigBee2Mqtt (« subscriber ») 34](#_Toc124502907)

[Annexe 14 –Programmation d’une application Android 34](#_Toc124502908)

[Références 34](#_Toc124502909)

[Langage Java 34](#_Toc124502910)

[Connexion au ZigBee2Mqtt (« subscriber ») 34](#_Toc124502911)

[Annexe 15 –Programmation d’une application iPhone 34](#_Toc124502912)

[Références 35](#_Toc124502913)

[Langage Swift 35](#_Toc124502914)

[Connexion au ZigBee2Mqtt (« subscriber ») 35](#_Toc124502915)

[Annexe 16 –Base de donnes 35](#_Toc124502916)

[Besoins 35](#_Toc124502917)

[Schéma 35](#_Toc124502918)

[Moteurs utilisés 35](#_Toc124502919)

[Procédures stockées (ou autre) 35](#_Toc124502920)

[Accès API 35](#_Toc124502921)

[Sécurité 35](#_Toc124502922)

# Équipe

## Identité

 Nom d’équipe : Plus Mario le professeur

## 

Logo / identifiant visuel

## Membres

|  |  |  |
| --- | --- | --- |
| **Nom** | **Responsabilités *(chef…)*** | **Contact (tel…)** |
| *David Pigeon* |  |  |
| *David Tremblay* |  |  |
| *Jean-Paul Bai* |  |  |
| *Florian Hostashy* |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 

# Demande

## Client

Alain Dubé,

Cégep de Rivière-du-loup

Représentant du Cégep de Rivière-du-Loup

80 rue Frontenac, Rivière-du-Loup Local : C-093

Téléphone : 418 862-6903 #2585

Courriel : MIO

Projet / Besoin

### Demande

Le département d'informatique de l’École du WEB aimerait avoir un système domotique pour ses nouveaux locaux. Vous devez dans le cadre de votre cours de « projets technologiques » implanter un système complet qui aura les fonctionnalités suivantes :

Au niveau technologique :

* L’utilisation d’un ou plusieurs Raspberry Pi
* L’utilisation de la technologie MQTT (mosquitto)
* Senseurs et actualisateurs
* Microcontrôleurs ESP32, arduino (au besoin)
* L’utilisation d’un « TFT Panel » SWTI de la compagnie Stone (au besoin)
* « Sniffer » ZigBee CC2531 (au besoin)

Au niveau fonctionnel :

* Pour l’équipe :
  + Avoir un broker fonctionnel
* Par membre :
  + Avoir une interface mobile **native** soit Android (Java) ou Apple (Swift)
  + Chaque application native devra avoir
    - Une gestion d’authentification (login)
    - Un système d’authentification IP pour se connecter au réseau local
    - L’internalisation
  + Avoir un actualisateur (différent des autres membres)
  + Avoir un senseur (différent des autres membres)
  + Avoir sa propre base de données pour sa gestion (API sécurisé)
  + Avoir sa propre logique applicative
  + Avoir un système de version (Git)
  + Document technique complet
  + Document(s) pour les usager(s)
  + Faire un plan de test opérationnel qui sera rempli par un coéquipier
  + Faire une remise officielle au client

# Conception

## Contexte

**Nom du client :** Cégep de Rivière-du-Loup

**Statut :** Éducation

**Mission :** Offrir un enseignement professionnel & guider les étudiants sur le marché du travail

**Services proposés :** Éducation Supérieure - Orientation – Divertissement

**Localisation :** 80 Rue Frontenac, Rivière-du-Loup, QC G5R 1R1

**Démographie :** ~1200 (Étudiants & personnel)

**Types d’utilisateurs :** Étudiants & personnel

**Infrastructure en place :**

* Réseau Wifi

**Plan des locaux :**



## Complément d’information

Visite des lieux, questionnement et interview (clients, usagers et partenaires).

## 

# Objectifs

1: Programmer le Broker - Raspberry Pi

2: Construire et programmer un ***senseur*** ESP32-MQTT-Wifi

3: Construire et programmer un ***actualisateur*** ESP32-MQTT-Wifi

4: Installer et configurer un ***senseur*** MQTT-ZigBee

5: Installer et configurer un ***actualisateur*** MQTT- ZigBee -RELAY

6: Programmation native

* Système de gestion d’accès (portes)
* Système de qualité de l’air
* Système d’alarme
* Information sur panneaux LEDs
* Alerte de température

## Objectif 1 : Programmer le Broker - Raspberry Pi

Spécifique

Installer et configurer un Broker Mosquitto dans un Raspberry Pi. L’objectif du broker est de gérer l’entrée et la sortie des données des différents senseurs. Avec l’aide de Mosquitto et du protocole MQTT, on reçoit les données des clients et les envoie aux « subscribers ».

Mesurable

Lorsque le broker reçoit une donnée d’un client « publisher » et l’envoie à un « subscriber » correctement.

Acceptable

Toutes les données reçues des senseurs sont envoyées à tous les « subscribers ». Nombre de senseurs <=10 et le nombre de « subscribers » <=10 dans l’environnement de l’École du WEB.

Réaliste

L’installation d’un broker MQTT est relativement facile. Plusieurs exemples existent sur Internet. De plus, le protocole est gratuit et peut facilement accueillir plusieurs dizaines de « publishers » et de « Subscribers ». On retrouve plusieurs tutoriels qui pourront aider à bien le configurer.

Temporellement défini

L’installation devrait prendre 2 heures/homme (après avoir fait une veille technologique complète) et devrait être un des premiers éléments à programmer.

## Objectif 2: Construire et programmer des senseurs ESP32-MQTT-Wifi

Spécifique

Ces senseurs devraient être programmés avec l’aide de contrôleurs ESP32. Ces derniers peuvent facilement se connecter à un Broker MQTT en mode « publisher » pour transmettre une valeur de clef extraite d’un senseur (RFID, gaz, températures…). La transmission peut se faire en utilisant le Wifi de l’infrastructure de l’École du WEB.

Le microcontrôleur suggéré est le ESP32:

|  |
| --- |
| Une image contenant équipement électronique, circuit  Description générée automatiquement[Esp32](https://www.amazon.ca/-/fr/KeeYees-d%C3%A9veloppement-Microcontr%C3%B4leur-ESP-WROOM-32-Arduino/dp/B07QCP2451)  Environ 12$ chacun |

Les senseurs sont déjà disponibles au CTEW. Il devra être installé sur le mur près du système de porte du C089.

Senseur 1 : La carte RFID

|  |
| --- |
| RFIS-[RC522](https://www.amazon.ca/RC522-RF-capteur-S50-Cartes-porte-cl%C3%A9s-Raspberry/dp/B07FFN54LS)  Environ 15$ chacun |

Senseur 2 : Le capteur de Gaz

|  |
| --- |
| [Une image contenant chapeau  Description générée automatiquement](https://www.aliexpress.com/item/1005001905705618.html?mp=1)Lots de capteurs de Gaz  *(MQ2 : Combustible gas, Smoke)*  Environ 20$ chacun  « [Amazon](https://www.amazon.ca/-/fr/Eiechip-capteurs-d%C3%A9tection-MQ-2-MQ-135/dp/B079NJML69)» |

Une image contenant texte

Description générée automatiquementSenseur 3 : Le capteur de CO2

|  |
| --- |
| [Une image contenant texte, équipement électronique  Description générée automatiquement](https://www.aliexpress.com/item/1005003028851389.html?mp=1) KEYESTUDIO CCS811  *Capteur C02*  Environ 20$ chacun  « [Amazon](https://www.amazon.ca/-/fr/KEYESTUDIO-D%C3%A9tecteur-dioxyde-Moniteur-temp%C3%A9rature/dp/B086HCSM6N)» |

Une image contenant texte

Description générée automatiquementSenseur 4 : Le capteur d’intensité

|  |
| --- |
| Une image contenant texte, équipement électronique, circuit  Description générée automatiquementGY-302 BH1750FVI  *Capteur Intensité*  Environ 10$ chacun  « [Amazon](https://www.amazon.ca/-/fr/508401/dp/B09HBCB8HF)» |

Une image contenant texte

Description générée automatiquementSenseur 5 : Le capteur de mouvement

|  |
| --- |
| Une image contenant intérieur, carrelé, sale  Description générée automatiquement HC-SR501  *Capteur mouvement*  Environ 10$ chacun  « [Amazon](https://www.amazon.ca/-/fr/d%C3%A9tection-mouvement-HC-SR501-PIR-Raspberry-Arduino/dp/B00NFWMW4C)» |

Mesurable / acceptable

Lorsque le broker reçoit une donnée (RFID ou autre selon le senseur en question) « publisher » avec les informations du senseur.

Réaliste

Tous les senseurs demandés sont compatibles avec le ESP32 et plusieurs tutoriels sont disponibles sur Internet.

Temporellement défini

La construction, l’installation, la programmation et la configuration dans le Broker devraient prendre 4 heures par senseurs et pourraient être réalisées après que le Broker est compris et bien configuré, soit à mi-parcours.

## Objectif 3: Construire et programmer des actualisateurs ESP32-MQTT-Wifi

Spécifique

Cet actualisateur devrait être programmé avec l’aide d’un contrôleur ESP32. Ce dernier peut facilement se connecter à un Broker MQTT en mode « receiver » pour recevoir une valeur de clef extraite d’un senseur (RFID, gaz, températures…). La réception peut se faire en utilisant le Wifi de l’infrastructure de l’École du WEB.

Le microcontrôleur suggéré est le ESP32:

|  |
| --- |
| Une image contenant équipement électronique, circuit  Description générée automatiquement[Esp32](https://www.amazon.ca/-/fr/KeeYees-d%C3%A9veloppement-Microcontr%C3%B4leur-ESP-WROOM-32-Arduino/dp/B07QCP2451)  Environ 12$ chacun |

Les actualisateurs sont déjà disponibles au CTEW. Il devra être installé dans l’environnement de l’École du WEB.

Actualisateur 1 : Écran OLED

|  |
| --- |
| OLED 128x64  Environ 15$ chacun |

Actualisateur 2 : Écran LCD

|  |
| --- |
| LCD 20x4  Environ 15$ chacun |

Actualisateur 3 : Écran LED

|  |
| --- |
| Led screen  Environ 100$ chacun |

Actualisateur 4 : Relay (5 ou 3.3V)

|  |
| --- |
| Relay  Environ 10$ chacun |

Actualisateur autre : le faire approuver par le client

|  |
| --- |
| ???  Environ ??$ chacun |

Mesurable / acceptable

Lorsque le broker envoie une donnée, l’actualisateur réagit selon la logique applicative voulue.

Réaliste

Tous les actualisateurs demandés sont compatibles avec le ESP32 et plusieurs tutoriels sont disponibles sur Internet.

Temporellement défini

La construction, l’installation, la programmation et la configuration dans le Broker devraient prendre 4 heures par senseurs et pourraient être réalisées après que le Broker est compris et bien configuré, soit à mi-parcours.

## Objectif 4 : Installer et configurer un **senseur** MQTT-ZigBee

Spécifique

Les senseurs Zigbee sont vendus et facilement configurables. Plusieurs sont compatibles avec zigbee2mqtt. C’est le cas du senseur de température et d’humidité SONOFF SNZB-02 et le bouton (switch) SNZB-01. Ils sont configurés en mode « publisher » et le «broker » devrait recevoir leurs données lorsqu’il sera bien configuré.

Nous suggérons les senseurs suivants :

|  |
| --- |
| Une image contenant texte, équipement électronique  Description générée automatiquement SONOFF SNZB-01  Environ 20$ chacun  [Amazon](https://www.amazon.ca/-/fr/dp/B08L4Y6SSW) |

|  |
| --- |
| SONOFF SNZB-02  Environ 20$ chacun  [Amazon](https://www.amazon.ca/SNZB-04-anti-effraction-s%C3%A9curit%C3%A9-compatible-incluses/dp/B08BFGJ6RB) |

Mesurable / acceptable

Lorsque le broker reçoit une donnée de température et/ou l’état du bouton.

Réaliste

Les éléments SONOFF sont compatibles avec les périphériques zigbee2mqtt et plusieurs informations sont disponibles sur Internet.

Temporellement défini

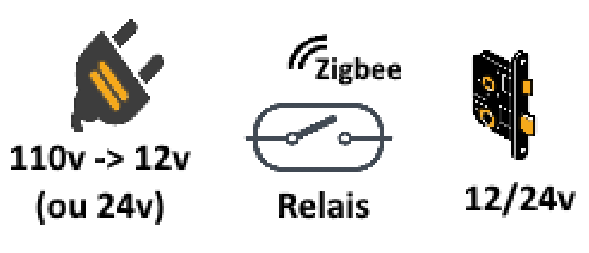
L’installation et la configuration dans le Broker devraient prendre 2 heures et pourraient être réalisées après que le Broker est compris et bien configuré, soit à mi-parcours.

## Objectif 5 : Installer et configurer un **actualisateur** MQTT- ZigBee -RELAY

Spécifique

Cet actualisateur ZigBee est un simple relai 110V qui permet de donner une charge à un appareil quelconque lorsque demandé. La commande est envoyée sous le format ZigBee et le dispositif est configuré comme un « subscriber » sous MQTT (en utilisant Zigbee2myqtt).

Dans cette infrastructure, il y deux actualisateurs Zigbee. Un pour permettre à la porte du C089 de s’ouvrir en activant le « sketch » de porte à distance. Il y a deux sortes de systèmes, un qui fonctionne avec le 12V et un autre sous 24V. Dans ce cas, un simple transformateur 110v/12v ou 110v/24v peut être utilisé pour activer le système. DEUX autres pour allumer une lumière dans le local CTEW (dompe) et dans le local C089.

 Une image contenant texte

Description générée automatiquement

Nous suggérons un des relais le suivant :

|  |  |
| --- | --- |
| Une image contenant équipement électronique, jack  Description générée automatiquement[SONOFF S31 Lite ZB](https://www.amazon.ca/-/fr/SONOFF-Zigbee-Fonctionne-SmartThings-n%C3%A9cessaire/dp/B08Y87WD1X?th=1)  Environ 20$ chacun | Une image contenant texte, fenêtre, blanc, appareil  Description générée automatiquement[SONOFF Basiczbr3](https://www.amazon.ca/-/fr/Basiczbr3-Controller-compatible-fonctionne-SmartThings/dp/B083DDYW9T)  Environ 20$ chacun |

Le système de porte est déjà installé dans l’infrastructure.

Mesurable

Lorsque le broker reçoit une donnée demandant l’ouverture du relai d’un client « publisher » et l’envoie à un relai zigbee « subscriber » correctement.

Acceptable

Le relai s’opère à chaque fois qu’un « publisher » le demande. C’est le cas pour l’ouverture de la porte et l’activation des lampes.

Réaliste

Les éléments SONOFF sont compatibles avec les périphériques zigbee2mqtt et plusieurs informations sont disponibles sur Internet.

Temporellement défini

L’installation et la configuration dans le Broker devraient prendre 2 heures et pourraient être réalisées après que le Broker est compris et bien configuré, soit à mi-parcours.

## Objectif 6 : Programmation native

Natif ou "cross-platform" ? Voilà la vraie question.

* [Développement mobile natif : que faut-il savoir ? (appstud.com)](https://www.appstud.com/fr/guides/agence-mobile/app068/)
* [Applications mobiles : développement natif ou technologie cross-platform ? On refait le match | AXOPEN](https://www.axopen.com/blog/2022/01/mobile-natif-crossplatform-comparatif/)
* [Application mobile hybride ou native : comment choisir ? - Codeur Blog](https://www.codeur.com/blog/application-mobile-hybride-ou-native/)

Programmer deux programmes (Android et iPhone) qui pourront transmettre et recevoir des données directement d’un broker « publisher » / « receiver ». Vous devez faire la gestion du senseur et de l’actualisateur selon une logique applicative donnée.

Vous devez programmer directement dans le langage Swift pour votre Apple et Java (ou Kotlin) pour l’Android.

Remarque : Bien qu’il est demandé d’utiliser un langage natif, vous pouvez utiliser un langage « hybride » qui génère du code natif. Vous ne pouvez pas utiliser un langage WEB.

1. Système de gestion d’accès (portes)



©[Image extraite d’Internet](https://www.youtube.com/watch?v=GOO84CGBPz8)

Spécifique

Le système de gestion de porte est un logiciel qui permet de faire la gestion des accès des portes à l’École du WEB. Le système devrait être conçu pour gérer plusieurs portes même s’il y en a seulement une dans notre infrastructure. Voici les caractéristiques du système :

* Une base de données qui contient les comptes, les adresses IP des périphériques, les usagers, les logs ainsi que les droits accès,
* Avoir une interface API/REST simple bien documenté et sécurisé,
* Partie MQTT « publisher » pour envoyer une commande d’ouverture de porte,
* Partie MQTT « subscriber » pour recevoir une demande d’accès,

Une image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquement Une image contenant texte, équipement électronique

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant objets métalliques, charnière

Description générée automatiquement

Demande « subscriber » « publisher » Ouverture

* Sections et fonctionnalités :
  + Authentification (login / logout),
  + Configuration
    - Wifi (adresse ip locale du broker extraite d’une base de données externe (API))
    - Adresse du broker
    - Liste des portes (avec les tags « subscriber » et « publisher »)
    - Nombre de secondes d’ouverture
    - Langage
  + Liste des dernières demandes d’accès
  + Liste et gestion des usagers (ajout, soustraction, droits d’accès aux portes …)
  + Liste des portes (ajout, soustraction …)
  + Commandes instantanées (demande d’ouverture …)

***D’autres fonctionnalités pourront être ajoutées en cours de mandat.***

Vous pouvez utiliser le format de la base de données que vous souhaitez, mais elle doit être bien documentée.

Mesurable

À chaque fois que le système reçoit une demande d’ouverture valide de porte par le Broker, l’application native envoie un message « publisher » au « broker » demandant l’ouverture de la bonne porte. La porte s’ouvre.

Acceptable

Toutes les fonctionnalités demandées sont fonctionnelles sur un téléphone Android (de votre choix) et/ou un téléphone iPhone (de votre choix).

Réaliste

L’application est relativement simple. La difficulté est plus sur le développement de l’application en natif. Plusieurs formations existent sur Internet. Ils vous faciliteront la tâche.

Temporellement défini

C’est probablement la partie la plus longue du projet. Cette tâche devrait être débutée dès la troisième semaine au minimum. Il faudrait prévoir plus de 30 heures. Un prototype devra être fonctionnel dès la semaine 5.

1. Système de qualité de l’air



©[Image extraite d’Internet](https://phys.org/news/2012-12-small-portable-sensors-users-exposure.html)

Spécifique

Le système de gestion de la qualité de l’air est un logiciel qui permet de connaitre la qualité de l’air à l’École du WEB. Le système devrait être conçu pour gérer plusieurs senseurs dans l’infrastructure. Voici les caractéristiques du système :

* Une base de données qui contient les comptes, les adresses IP des périphériques, les usagers, les logs ainsi que les droits accès,
* Avoir une interface API/REST simple bien documenté et sécurisé,
* Partie MQTT « publisher » pour envoyer une commande d’ouverture de porte,
* Partie MQTT « subscriber » pour recevoir une demande d’accès,

Une image contenant texte, projecteur

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquement Une image contenant texte, équipement électronique

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquement

Données « subscriber » « publisher » Affichage

* Sections et fonctionnalités :
  + Authentification (login / logout),
  + Configuration
    - Wifi (adresse ip locale du broker extraite d’une base de données externe (API))
    - Adresse du broker
    - Liste des senseurs (avec les tags « subscriber » et « publisher »)
    - Nombre de secondes avant d’envoyer une donnée au BROKER
    - Langage
  + Liste des dernières données
  + Liste et gestion des senseurs (ajout, soustraction …)
  + Liste et gestion des actualisateurs (ajout, soustraction …)
  + Commandes instantanées (alertes …)

***D’autres fonctionnalités pourront être ajoutées en cours de mandat.***

Vous pouvez utiliser le format de la base de données que vous souhaitez, mais elle doit être bien documentée.

Mesurable

À chaque fois que le système reçoit une donnée valide du Broker, l’application native envoie un message « publisher » au « broker » demandant l’affichage de la qualité de l’air. L’actualisateur affiche la bonne donnée et/ou l’alerte.

Acceptable

Toutes les fonctionnalités demandées sont fonctionnelles sur un téléphone Android (de votre choix) et/ou un téléphone iPhone (de votre choix).

Réaliste

L’application est relativement simple. La difficulté est plus sur le développement de l’application en natif. Plusieurs formations existent sur Internet. Ils vous faciliteront la tâche.

Temporellement défini

C’est probablement la partie la plus longue du projet. Cette tâche devrait être débutée dès la troisième semaine au minimum. Il faudrait prévoir plus de 30 heures. Un prototype devra être fonctionnel dès la semaine 5.

1. Système d’alarme

 Une image contenant texte, tableau de points

Description générée automatiquement

Spécifique

Le système d’alarme est très classique en soi. Il détecte la présence d’une personne, la lumière ou l’ouverture d’une porte à l’École du WEB. Le système devrait être conçu pour gérer plusieurs senseurs installés dans l’infrastructure. Lorsqu’une présence est détectée, l’application transmet, par la voie du broker, l’alerte à l’écran ainsi que le code (NIP) qu’il faut entrer afin de désactiver l’alerte. Voici les caractéristiques du système :

* Une base de données qui contient les comptes, les adresses IP des périphériques, les usagers, les logs ainsi que les droits accès,
* Avoir une interface API/REST simple bien documenté et sécurisé,
* Partie MQTT « publisher » pour envoyer une commande d’ouverture de porte,
* Partie MQTT « subscriber » pour recevoir une demande d’accès,

Une image contenant texte

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquement Une image contenant texte, équipement électronique

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant texte, tableau de points

Description générée automatiquement

Présence « subscriber » « publisher » Affichage

* Sections et fonctionnalités :
  + Authentification (login / logout),
  + Configuration
    - Wifi (adresse ip locale du broker extraite d’une base de données externe (API))
    - Adresse du broker
    - Liste des senseurs (avec les tags « subscriber » et « publisher »)
    - Nombre de secondes avant d’envoyer une donnée au BROKER
    - Langage
  + Liste des dernières données
  + Liste et gestion des senseurs (ajout, soustraction …)
  + Liste et gestion des actualisateurs (ajout, soustraction …)
  + Commandes instantanées (alertes …)

***D’autres fonctionnalités pourront être ajoutées en cours de mandat.***

Vous pouvez utiliser le format de la base de données que vous souhaitez, mais elle doit être bien documentée.

Mesurable

À chaque fois que le système reçoit une donnée valide du Broker, l’application native envoie un message « publisher » au « broker » demandant l’affichage de l’alerte ainsi que le NIP. L’actualisateur affiche l’alerte jusqu’à ce que l’usager affiche le bon code.

Acceptable

Toutes les fonctionnalités demandées sont fonctionnelles sur un téléphone Android (de votre choix) et/ou un téléphone iPhone (de votre choix).

Réaliste

L’application est relativement simple. La difficulté est plus sur le développement de l’application en natif. Plusieurs formations existent sur Internet. Ils vous faciliteront la tâche.

Temporellement défini

C’est probablement la partie la plus longue du projet. Cette tâche devrait être débutée dès la troisième semaine au minimum. Il faudrait prévoir plus de 30 heures. Un prototype devra être fonctionnel dès la semaine 5.

1. Information sur panneaux LEDs

Spécifique

Le système d’information est très classique en soi. Il permet d’envoyer des commandes à un écran LED installé à l’École du WEB. Le système devrait être conçu pour gérer plusieurs écrans LED installés dans l’infrastructure. De plus, l’application peut envoyer des commandes telles que s’allumer, s’éteindre, s’ouvrir ou s’allumer à des temps définis. Voici les caractéristiques du système :

* Une base de données qui contient les comptes, les adresses IP des périphériques, les usagers, les logs ainsi que les droits accès,
* Avoir une interface API/REST simple bien documenté et sécurisé,
* Partie MQTT « publisher » pour envoyer une commande d’ouverture de porte,
* Partie MQTT « subscriber » pour recevoir une demande d’accès,

Une image contenant texte, personne, équipement électronique, main

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquement Une image contenant texte, équipement électronique

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant flèche

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Commande « subscriber » « publisher » Affichage

* Sections et fonctionnalités :
  + Authentification (login / logout),
  + Configuration
    - Wifi (adresse ip locale du broker extraite d’une base de données externe (API))
    - Adresse du broker
    - Liste des senseurs (avec les tags « subscriber » et « publisher »)
    - Nombre de secondes avant d’envoyer une donnée au BROKER
    - Langage
  + Liste des dernières données
  + Liste et gestion des senseurs (ajout, soustraction …)
  + Liste et gestion des actualisateurs (ajout, soustraction …)
  + Commandes instantanées (veille, ouverture, fermeture, alertes …)

***D’autres fonctionnalités pourront être ajoutées en cours de mandat.***

Vous pouvez utiliser le format de la base de données que vous souhaitez, mais elle doit être bien documentée.

Mesurable

À chaque fois que le système reçoit une donnée valide du Broker, l’application native envoie un message « publisher » au « broker » demandant l’affichage de l’alerte et/ou de messages.

Acceptable

Toutes les fonctionnalités demandées sont fonctionnelles sur un téléphone Android (de votre choix) et/ou un téléphone iPhone (de votre choix).

Réaliste

L’application est relativement simple. La difficulté est plus sur le développement de l’application en natif. Plusieurs formations existent sur Internet. Ils vous faciliteront la tâche.

Temporellement défini

C’est probablement la partie la plus longue du projet. Cette tâche devrait être débutée dès la troisième semaine au minimum. Il faudrait prévoir plus de 30 heures. Un prototype devra être fonctionnel dès la semaine 5.

# Veille technologique

Inclure vos recherches sur les technologies. Faire une **synthèse** des éléments importants ici et tout le reste dans l’annexe correspondante.

## Raspberry Pi (voir annexe 1)

## Mqtt (voir annexe 2)

## ZigBee2MQTT (voir annexe 3)

## Senseur RFiD-ESP32 (voir annexe 4)

## Senseur de présence-ESP32 (voir annexe 5)

## Senseur de luminosité-ESP32 (voir annexe 6)

## Senseur de CO2-ESP32 (voir annexe 7)

## Senseur de Gaz-ESP32 (voir annexe 8)

## Senseur de température-Zigbee (voir annexe 9)

## Senseur switch-Zigbee (voir annexe 10)

## Écran Stone (voir annexe 11)

## Contrôleur d’ouverture de porte Zigbee (voir annexe 12)

## Contrôleur de lumière Zigbee (voir annexe 13)

## Programmation d’Android native (voir annexe 14)

## Programmation iPhone native (voir annexe 15)

## Base de données (voir annexe 16)

# Stratégies de développement

## Répartition des tâches

Chaque équipe doit séparer les tâches entre les membres. Chaque membre choisit une lettre dans le tableau qui suit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Éléments** | **Choix** | | | | |
|  | **A** | **B** | **C** | **D** | **E** |
| Programmation |  |  |  |  |  |
| Raspberry Pi 4 – MQTT – ZigBee2MQTT | x | | | | |
| Bases de données / Api (NodeJS/MongoDB) | x | x | x | x | x |
| Programmation d’une application Android/iPhone | x | x | x | x | x |
|  |  |  |  |  |  |
| Programmation d’une application Android/iPhone \* |  |  |  |  |  |
| Système d’ouverture porte | x |  |  |  |  |
| Senseur RFid |  |  |  |  |  |
| Actualisateur ouverture porte |  |  |  |  |  |
| Système de qualité de l’air |  | x |  |  |  |
| Senseur (Gaz, Co2, température) |  |  |  |  |  |
| Actualisateur (Oled ou lumière) |  |  |  |  |  |
| Système d’alarme |  |  | x |  |  |
| Senseur (Présence, luminosité, détecteur d’ouverture) |  |  |  |  |  |
| Actualisateur (Écran Stone) |  |  |  |  |  |
| Information sur panneaux LED |  |  |  | x |  |
| Senseur (présence) |  |  |  |  |  |
| Actualisateur : LEDS |  |  |  |  |  |
| À définir |  |  |  |  | x |
| Senseur (à définir) |  |  |  |  |  |
| Actualisateur : (à définir) |  |  |  |  |  |

*\* Il est possible de programmer un autre système, mais il faut le faire approuver par le client*

## Organisation du travail

* + - Chaque équipier est responsable des tâches qui lui sont distribuées.
    - La correction, la notation et le suivi se font INDIVIDUELLEMENT.
    - Chaque équipe est responsable du bon fonctionnement global du projet.
    - Il n’est pas possible de redistribuer les tâches.
    - Les tâches peuvent être modifiées seulement avec l’accord du demandeur (client)
    - Les équipiers ont droit de s’entraider s’ils sont dans la même équipe, évidemment.
    - Les équipes ne peuvent pas s’entraider. Cependant, il est possible de demander une (des) question(s) à l’autre équipe qui devra répondre au mieux de ses connaissances. Ce n’est pas une compétition, mais bien une occasion d’apprendre.
    - L’enseignant fournira du matériel et des ressources afin de faciliter le travail.
    - L’enseignant sera présent et aura le rôle de facilitateur en cours de session.

***Il est important de spécifier que lors de la remise 1, vous devez obligatoirement avoir un prototype fonctionnel avec tous les morceaux.***

### Outils de partage (collaboration)

L’enseignant partagera une session Microsoft 360 Team.

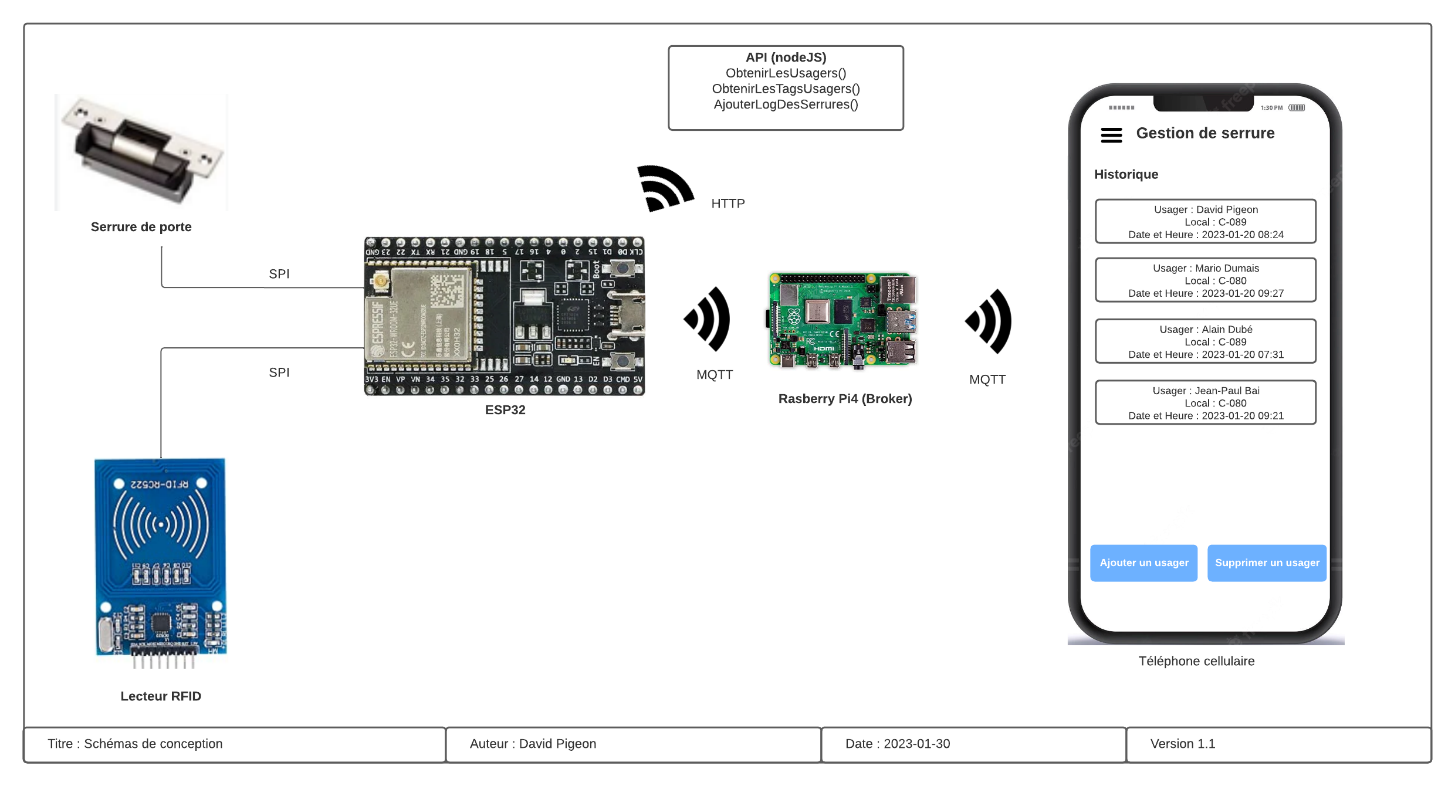
### Méthode de gestion de projet utilisée

Il est fortement suggéré d’utiliser une méthode Agile entre les membres de l’équipe.

### Gestion des versions

Chaque programme devra avoir son GIT actualiser aux semaines (minimum).

## Schéma et explications



## Calendrier des livrables

Vous ne pouvez pas changer les éléments en bleu déjà inscrits dans ce calendrier des livrables. Écrire vos éléments en vert.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Date*** | ***Action* /objectifs** | ***Tâches*** | ***Réalisées*** |
| **19 et 20 janvier** | Prendre connaissance du projet  Détermination des tâches à réaliser  Diagramme de conceptions  Configuration de l’API  Veille technologique de Kotlin | Configurer l’API en NodeJS et metre la base de données en MongoDB.  Effectuer des recherches pour Kotlin | x |
| **26 et 27 janvier** | Remise des tâches à accomplir et votre veille technologique.  Ajustement des tâches à réaliser  Début du développement de l’application (backend)  L’ESP32 doit pouvoir récupérer les informations de l’API et de comparer le code obtenu par le RFID par les informations récupérées dans la bd | Commencer la programmation sur l’ESP32 (Configuration du senseur RFID) | x |
| **2 et 3 février** | Programmer l’application mobile pour qu’il puisse parler avec le broker | Apprendre Kotlin et commencer à programmer pour que l’application puisse parler avec le broker | **x** |
| **9 et 10 février** | Rendre le relais avec l’ESP fonctionnel | Continuer à programmer dans l’application native pour que celui-ci fonctionne  Programmer l’ESP pour qu’il puisse faire des actions au relais | **x** |
| **16 et 17 février** | Remise 1, démonstration et évaluation du projet (livrable)  Prototype 1  Voir dans le plan de cours | Faire la logique applicative de l’application pour récupérer les informations des tags et d’autorisé ou pas à ouvrir certaine serrure. | **x** |
| **23 et 24 février** | Enregistrer le scan des tags dans une table de logs | Faire en sorte que le système envoie le tag scanner dans une table de logs |  |
| **16 et 17 mars** | Afficher la table de logs sur l’application native | Programmer l’application pour qu’il puisse faire des appels d’API |  |
| **23 et 24 mars** | Faire un login pour les utilisateurs | Programmer un login dans l’application native  Faire un appel d’API sécurisé pour les logins |  |
| **30 et 31 mars** | Lier chacun des modules ensembles | Faire le branchement des différents modules  Programmer pour que tous les modules fonctionnent ensemble |  |
| **6 et 13 avril** |  | Continuer la programmation pour lier chacun des modules    Temps alloué pour le débogage |  |
| **14 et 20 avril** | Remise 2, démonstration et évaluation du projet (livrable)  Voir dans le plan de cours | Continuer la documentation technique |  |
| **21 et 27 avril** | Compléter la documentation | Continuer la documentation  Faire un Readme pour le GitHub |  |
| **28 avril et 4 mai** | Ajustement dans le code | Faire de la revue de code |  |
| **5 et 11 mai** | Ajustements  Faire remplir votre plan de test fonctionnel par un coéquipier | Compléter le plan de test fonctionnel d’un autre coéquipier |  |
| **19 et 25 mai** | Remise officielle au client avec tous les documents  Évaluation finale |  |  |

## Mise en œuvre

**L’application sera disponible sur GitHub avec différente version. Chaque version ajoute des fonctionnalités à l’application et corrige les problèmes que l’application peut contenir.**

1. **Prise en compte du projet**

L’objectif du projet est de réaliser un système permettant de faire la gestion d’ouverture et de fermeture d’une serrure. Une application mobile devra être disponible pour avoir une interface pour l’usager afin qu’il puisse faire la gestion des serrures.

1. **Faire l’analyse fonctionnel**

Pour réaliser ce système nous aurons besoins de plusieurs éléments qui devront communiquer entre eux;

- Un microcontrôleur (ESP32)

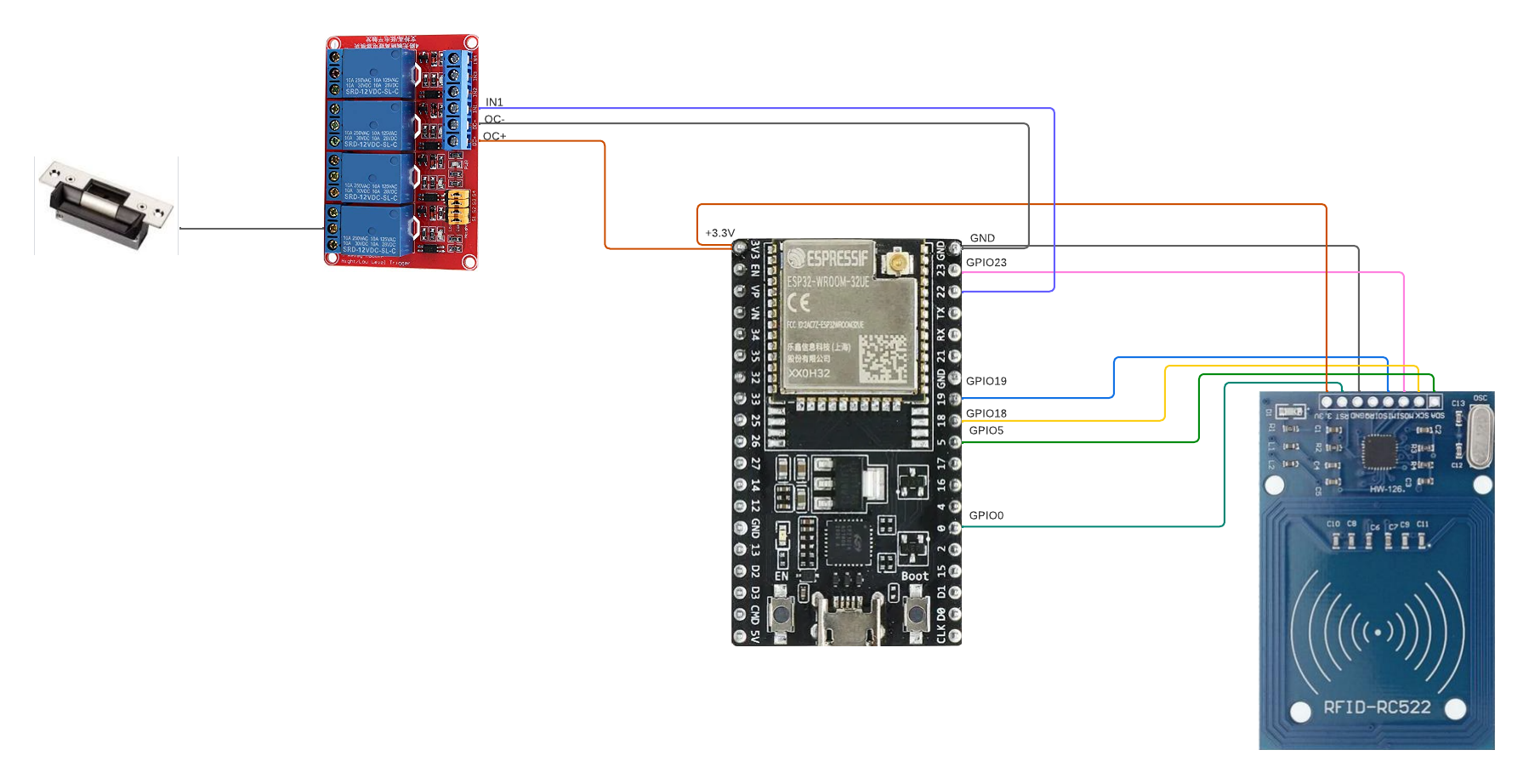
- Un broker (Raspberry Pi4)

- Une API (VPS OVH en nodeJS)

- Téléphone mobile (Android)

Les éléments mentionner ci-dessus seront directement lier ensemble pour s’envoyer de l’information. Le téléphone pourra obtenir/envoyer de l’information au broker pour qu’il soit obtenu/envoyer au microcontrôleur grâce au protocole MQTT. L’API quant à lui, permettra de communiquer avec la base de données et avec le téléphone grâce au protocole HTTP.

1. **Faire le schéma de branchement**



[Format PDF](https://cegeprdlca.sharepoint.com/:b:/r/sites/Projetstechnologiques420-009-RL-DavidPigeon/Documents%20partages/%C3%89quipe%204%20-%20David%20Pigeon%20(portes)/Sch%C3%A9maDeBranchment.pdf?csf=1&web=1&e=sF3hRd)

1. **Configuration de l’API**

L’API sera programmée en NodeJS. Il a bien évidemment énormément de language que nous aurions pu utiliser, mais NodeJS à l’avantage d’être extrêmement facile à déployer sur un serveur. Notre base de données sera en MongoDB. Nous utiliserons donc l’ORM Mongoose pour faire nos liens entre l’API et la base de données. Un Bearer Token sera mis en place pour sécuriser notre API. Les appels d’API retourneront du JSON qui sera ensuite traité dans l’application. Des requêtes en GET seront implémentées pour l’obtention des données. Des requêtes en POST permettront de faire le login de l’utilisateur.

1. **Programmation du système sur l’ESP**

L’ESP aura une configuration Wifi grâce à la librairie “Wifi.h”. De plus, une configuration de point d’accès sera aussi disponible pour permettre à l’utilisateur de choisir le réseau utilisé grâce à une page web. Celui-ci pourra aussi configurer son adresse IP en statique directement à partir de l’interface web.

Un module MQTT sera aussi intégré à l’ESP. Ce module permettra à l’ESP de pouvoir envoyer/recevoir des données à partir du Broker. On utilise donc la librairie “PubSubClient.h” qui permet de se connecter à un Topic sur notre Broker et d’être compatible avec le protocole MQTT.

La gestion des états est crucial pour l’ESP. C’est pour cela que plusieurs variables seront assignées pour faire cette gestion. Par exemple, lorsqu’on reçoit une donnée du broker, la variable permettant de faire la gestion de l’état pour la réception des données sera mis à “true”. Cela permet d’améliorer les performances de l’ESP étant donné que la mémoire de celui-ci est très limitée. Nous devons s’assurer que tout soit stable pour éviter de futurs problèmes.

1. **Programmation de l’application native**

L’application native se fera sur Kotlin qui est un dérivé du Java. Kotlin permet de s’simplifier le Java en retirant au maximum les éléments de structure ou de logique applicative. Ce langage offre énormément de librairie pour simplifier la programmation. Ces librairies sont sous forme de “gradle” qui permette d’aller chercher directement du code source pour pouvoir ensuite l’utiliser dans notre code. Il faut tout de même faire attention, car certain “gradle” ne sont plus à jour selon la version de Kotlin. Par ce fait, il faut s’assurer que nous prenons la bonne version selon celle de Kotlin.

Pour chaque module, une librairie y sera associée ([Liste des “gradles” utilisés](https://github.com/DavidPcegep/KotlinPorteSolace/blob/main/app/build.gradle)).

Cependant, quelques défauts sont à noter pour la programmation avec Kotlin.

* Kotlin est un langage qui gère très mal les valeurs Null. Il est donc important que toutes les valeurs instanciées n’aient pas de valeurs Null. Comparativement au langage C/C++ ou l’on peut définir une valeur Null grâce à la propriété NULL ou ptrnull (Pour définir un pointeur vide).
* Les “gradles” peuvent être une grande source de problème. Il faut s’assurer que ceux-ci ne soient pas nécessairement mis à jour, mais bien qu’il soit stable dans l’application. Si l’application est fonctionnelle avec les “gradles” actuels, il ne faut pas les m’être à jour.
* Certaines versions d’Android ne peuvent pas utilisées l’application. Il faut donc définir manuellement une version minimum pour que l’application soit le plus stable possible.
* Le débogage peut être une tâche quand l’application plante. Il faut savoir et comprendre notre code pour s’assurer que l’on comprenne d’où vient l’erreur.

1. **Compléter la documentation**
2. **Revue de code**
3. **Test fonctionnel**
4. **Remise du projet**

## Références et documentation

**Le lien vers GitHub :** [**https://github.com/DavidPcegep/ESP32\_Porte**](https://github.com/DavidPcegep/ESP32_Porte)

[**https://github.com/DavidPcegep/KotlinPorteSolace**](https://github.com/DavidPcegep/KotlinPorteSolace)

[**https://github.com/DavidPcegep/ApiPorte**](https://github.com/DavidPcegep/ApiPorte)

## Liste des versions (Gits)

|  |  |  |  |
| --- | --- | --- | --- |
| ***Nom*** | ***Version*** | ***Description*** | ***Lien sur GitLab/GitHub*** |
| **Version 1 de l’application mobile** | **V1.0** | **L’application est fonctionnelle. Elle peut communiquer avec l’API, et avec le broker. Un login est mis en place et on peut consulter l’historique des ouvertures de serrures.** | [**https://github.com/DavidPcegep/KotlinPorteSolace/releases/tag/V1.0**](https://github.com/DavidPcegep/KotlinPorteSolace/releases/tag/V1.0) |
| **Version 1 du microcontrôleur** | **V1.0** | **Le système est fonctionnel. On peut scanner un tag et envoyer le numéro du tag dans un topic dans le broker. On peut recevoir une information du broker et déverrouiller la serrure.** | [**https://github.com/DavidPcegep/ESP32\_Porte**](https://github.com/DavidPcegep/ESP32_Porte) |
| **Version 1 de l’API** | **V1.0** | **L'API est fonctionnelle. On peut envoyer et recevoir de l'information pour l'application mobile** | [**https://github.com/DavidPcegep/ApiPorte**](https://github.com/DavidPcegep/ApiPorte) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Annexes – Veilles technologiques

## Annexe 1 –Raspberry Pi

### **Caractéristiques**

Un Raspberry Pi est un ordinateur de petite taille qui est utilisé pour diverses applications telles que la programmation, l'électronique, les projets de robotique, la création de media center, et bien d'autres encore. Il est basé sur un système d'exploitation Linux et est doté de différents ports tels que HDMI, USB, Ethernet, entre autres pour connecter des périphériques externes. Il est également équipé d'un processeur et de mémoire, ce qui lui permet de fonctionner de manière indépendante. Il est très populaire auprès des développeurs, des étudiants et des passionnés de la technologie car il est abordable et facile à utiliser. En utilisant un Raspberry Pi, vous pouvez apprendre les bases de la programmation, de l'électronique et de l'informatique tout en réalisant vos propres projets créatifs.

Processeur : Un processeur Arm Cortex-A72 quad-core à 64 bits cadencé à 1,5 GHz

Mémoire vive (RAM) : 2 Go, 4 Go ou 8 Go de LPDDR4-3200 SDRAM (en fonction du modèle)

Stockage : Prise en charge de la mémoire flash microSD pour le stockage du système d'exploitation et des données

Connectivité : Gigabit Ethernet, 802.11ac Wireless, Bluetooth 5.0, BLE, 2 ports USB 3.0, 2 ports USB 2.0, 2 ports micro-HDMI (1,4a jusqu'à 4Kp60 pris en charge), 3,5 mm audio jack et composite vidéo, 40 broches GPIO, 2 ports pour caméra, 2 ports pour écran

Sortie vidéo : Prise en charge de la résolution jusqu'à 4Kp60

Alimentation : Alimentation par USB-C (minimum 3 A)

Dimensions : 88 x 58 x 19,5 mm

Température de fonctionnement : 0–50 °C

Ces caractéristiques font du Raspberry Pi 4 un ordinateur compact et puissant qui peut être utilisé pour des applications telles que la programmation, l'électronique, les projets de robotique, la création de media center, et bien d'autres encore.

### **Installation de l’OS**

Téléchargez l'image de Raspbian depuis le site officiel de Raspberry Pi.

Utilisez un programme comme balenaEtcher pour graver l'image sur une carte microSD.

Insérez la carte microSD dans le Raspberry Pi et démarrez-le.

Suivez les étapes de configuration initiale, telles que la configuration de la langue, du clavier, de l'utilisateur, etc.

Une fois la configuration initiale terminée, vous devriez être dirigé vers l'écran d'accueil de Raspbian.

Configuration et installation dans l’environnement de l’ÉDW

## Annexe 2 –Mqtt

### Technologie

MQTT est un protocole de communication utilisé pour connecter des appareils ensemble en utilisant un système de Publisher/Subscriber. Les appareils connectés (appelés clients) peuvent publier des informations sur des sujets spécifiques, et d'autres appareils qui sont abonnés à ces sujets peuvent recevoir ces informations. Cela permet une communication efficace en ne transmettant des informations qu'aux appareils qui sont intéressés par elles. Il est souvent utilisé pour les réseaux de capteurs et les objets connectés car il est efficace pour la consommation d'énergie et la bande passante.

### Installation d’un broker dans un Raspberry Pi

### Installation d’une fonctionnalité « subscriber » et « publisher »

#### Dans un Raspberry Pi (Linux C++)

#### Dans en ESP32

#### Dans iPhone (Swift)

#### Dans Android (Java)

## Annexe 3 –ZigBee2MQTT

### Technologie

### Installation dans un Raspberry Pi

### Utilisation d’un « Sniffer » Zigbee CC2531

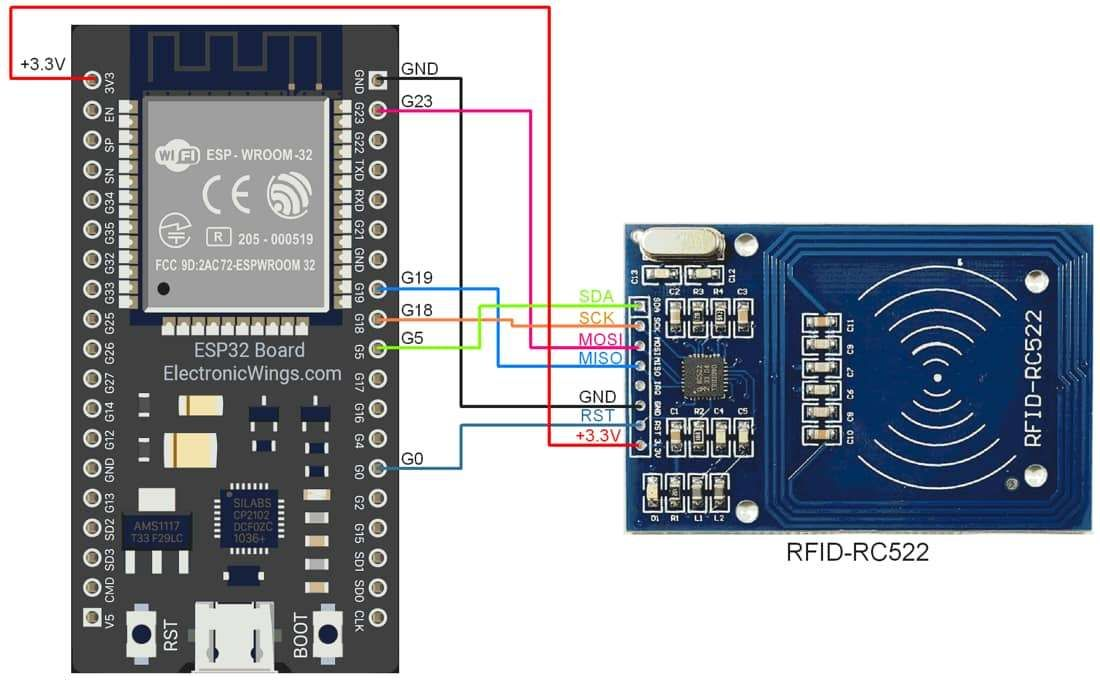
## Annexe 4 –Senseur RFId-Esp32

### Références

[electronicwings](https://www.electronicwings.com/esp32/rfid-rc522-interfacing-with-esp32)

### Schéma explicatif

### Schéma de branchement



### Programmations

Voir sur :

<https://github.com/DavidPcegep/ESP32_Porte/blob/dev/src/main.cpp>

<https://github.com/DavidPcegep/ESP32_Porte/blob/dev/src/MyRfid.h>

<https://github.com/DavidPcegep/ESP32_Porte/blob/dev/src/MyRfid.cpp>

### Libraires utilisées

# MFRC522 par Miguel André Balboa

### Connexion au Mqtt (« publisher »)

### Difficultés et solutions

### Documentation technique

### Bilan / conclusion

## Annexe 12 –Contrôleur de porte Zigbee

### Références

### Schémas

### Programmations

### Connexion au ZigBee2Mqtt (« subscriber »)

## Annexe 13 –Contrôleur de lumière Zigbee

### Références

### Schémas

### Programmations

### Connexion au ZigBee2Mqtt (« subscriber »)

## Annexe 14 –Programmation d’une application Android (Kotlin)

### Références

Il existe de nombreuses références pour apprendre Kotlin, comme le site officiel de la documentation de Kotlin, les tutoriels de la communauté Kotlin sur Medium, ainsi que des livres tels que "Kotlin in Action" de Dmitry Jemerov et Svetlana Isakova.

### Langage Kotlin (Java)

Kotlin est un langage de programmation open-source développé par JetBrains qui est de plus en plus utilisé pour développer des applications Android. Il est compatible avec le langage Java, ce qui facilite la migration des projets existants vers Kotlin.

### Connexion au ZigBee2Mqtt (« subscriber »)

En ce qui concerne la connexion à ZigBee2Mqtt, il existe des bibliothèques Kotlin telles que "kotlin-mqtt" qui peuvent faciliter la mise en place d'un subscriber pour se connecter à un serveur MQTT et recevoir des messages ZigBee2Mqtt. Il existe également des exemples de projets open-source développés en Kotlin qui intègrent ZigBee2Mqtt, comme "zigbee2mqtt-android" qui est une application Android qui permet de configurer et de gérer un réseau Zigbee via ZigBee2Mqtt.

## Annexe 15 –Programmation d’une application iPhone

### Références

### Langage Swift

### Connexion au ZigBee2Mqtt (« subscriber »)

## Annexe 16 –Base de données

### Besoins

Le système doit être capable d’appeler du contenu dans une API. Ce contenu doit être en format JSON.

### Schéma

### Moteurs utilisés

### Procédures stockées (ou autre)

### Accès API

### Sécurité