Rapport de projet

Projet Multidisciplinaire 3

Présenté par **Guns Antoine**

Année académique 2021-22

TECHNIQUE

Table des matières

1 Introduction

Ce projet regroupe 3 cours: Anglais 4, Système embarqué 2 et développement mobile, projet multidisciplinaire 3. Également, contrairement aux autres années, ce projet est individuel.

La tache demandée est la suivante : un programme sur un esp8366 doit permettre de scanner une carte RFID et en fonction de si celle-ci possède un droit d'accès, allumer différentes leds et afficher le numéro de la carte sur un afficheur 7 segments. Pour connaitre les droits d'accès des cartes ainsi que leur numéro, l'esp 8266 doit aller les chercher dans une base de données.

Cette base de données contient le tag des différentes cartes, le numéro et le droit d'accès. Mais également les logs des différentes cartes qui ont été scannées par le lecteur RFID, quel que soit leur accès.

Quant à l'application Android, celle-ci doit permettre le visionnage des logs des cartes mais également de pouvoir modifier les droits d'accès des cartes dans la base de données.

Ce projet représente assez bien la réalité sur le terrain car beaucoup d'entreprises utilisent un système de badge ou de carte d'accès dans leurs locaux et chaque lecteur RFID est relié à une même base de données pour permettre une centralisation des permissions et des logs.

Ce projet fut très intéressant à mener, de la création du programme pour l'esp en passant par la création de la base de données et le fonctionnement de l'application Android.

2 Cahier des charges

2.1 Projet

Système embarqué

Quand une carte RFID est scannée le système autonome doit…

- Détecter et identifier le tag RFID
- Se connecter à la base de données utilisant une connexion wifi
- Afficher le nombre de la carte scannée sur son afficheur 7 segments
- Allumer des leds en fonction des droits d'accès : vert autorisé, rouge refusé.

Développement mobile

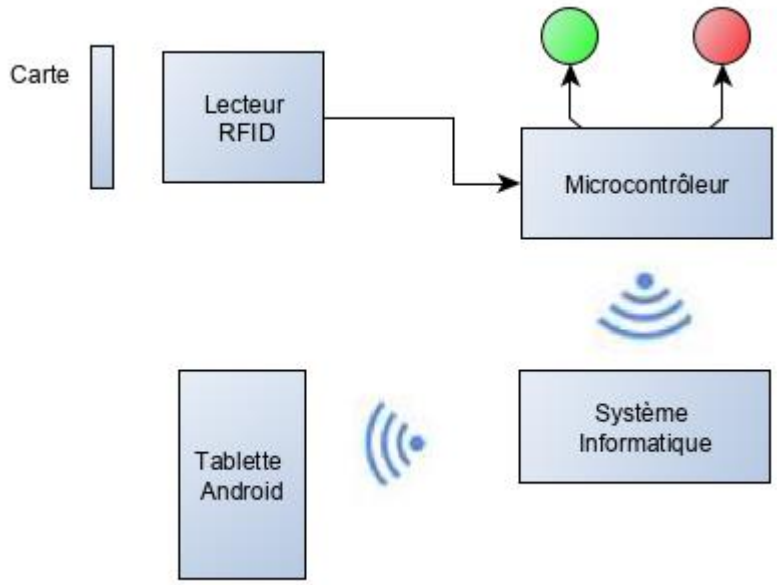
- Une page de login permettant de pouvoir accéder à l'application
- Une page permettant la visualisation des logs
- Une page permettant de modifier le droit d'accès de n'importe quelle carte.

Anglais

- Parler entre élèves et professeurs en anglais

- Faire une présentation du projet en anglais
- Ecrire et développer un résumé argumentatif personnel contenant :
    - un résumé général du projet,
    - le point de vue personnel comme si on avait dû le rajouter dans le portfolio (qu'a-t-on appris, quelles sont les choses à améliorer niveaux techniques ?).

2.2 Schéma du système



3 *Base de données et middleware*

3.1 Introduction

La base de données est le cœur du système car celle-ci procure à l'esp mais également à l'application Android les ressources et l'information nécessaires pour le bon fonctionnement du projet. Le Middleware, quant à lui, permet d'effectuer une liaison entre, d'un côté, la dB et l'esp et, de l'autre, entre la dB et l'application Android. Celui-ci est constitué de divers fichiers PHP.

Sans la base de données et le middleware, les 2 autres parties seraient complètement inutiles et ne pourraient fonctionner .

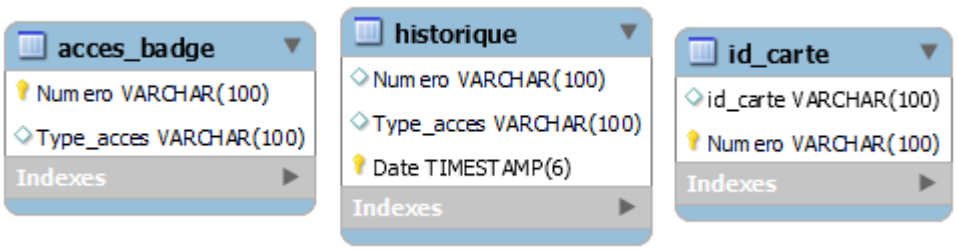
3.2 Hébergement

Pour pouvoir héberger la base de données, on utilise WAMPSERVER, celui-ci est gratuit et très simple d'utilisation. Il permet d'héberger tout ce qu'il faut pour la base de données et également de créer des redirections sur certaines adresses IP ou encore d'héberger des sites web avec Apache.

3.3 Base de données

La base de données est en MySQL, celle-ci est assez simple car elle ne contient que 3 tables : 1 pour stocker le numéro des cartes ainsi que leur tag, une autre pour stocker le numéro des cartes ainsi que leur accès et la dernière stockera les logs des cartes contenant leur numéro, accès au moment du scan ainsi que la date et l'heure du dit scan.

Schéma UML:



3.4 Middleware

Celui-ci est composé de plusieurs fichiers PHP, ceux-ci permettent de faire la liaison entre la dB, l'esp et l'Android. Ils sont hébergés sur le serveur Apache de WAMP et atteignables grâce à une adresse IP précise (grâce à une redirection).

Ceux-ci sont de 2 types. Les premiers : les fichiers d'envoi permettent d'apporter des modifications à la base de données que ce soit de simples modifications ou des ajouts d'éléments.

Les seconds : les fichiers de collecte, quant à eux, permettent simplement de lire la base de données.

Le début de chaque fichier PHP est le même :

```php
$dbname = 'arduino';
$dbuser = 'root';
$dbpass = '1234';
$dbhost = 'localhost';
$MDP = 'EZqVtcX4571';
```

On a quelques variables pour la connexion à la base de données (les 4 premières) celles-ci regroupent le nom de la base, l'utilisateur et le mot de passe pour la connexion ainsi que l'host (ici localhost car la base de données est en local).

On a également une variable MDP, celle-ci contient un mot de passe.

```php
$mdp= $_GET["Mdp"];
if (strcmp($MDP,$mdp) == 0){

    $connect = @mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);

    if(!$connect){
        echo "Error: " . mysqli_connect_error();
        exit();
    }
```

Celui-ci est utilisé quand la tablette ou l'esp utilisent un fichier PHP. Ils fournissent un mot de passe et celui-ci est comparé à celui de notre variable. Si les 2 sont identiques alors le fichier continue et on se connecte aux dB. Sinon on ne fait rien. On le récupère de l'url avec $_GET["Nom"], cette petite fonction permet de récupérer une information dans l'url portant le même nom que ce qu'on lui donne en paramètre. Si la connexion à la dB a échoué, alors le fichier s'arrête.

3.5 *Fichiers d'envoi*

```php
$Num= $_GET["Num"];
$Acces = $_GET["Acces"];
$query = "INSERT INTO `historique` (`Numero`, `Type_acces`, `Date`) VALUES ('$Num', '$Acces', CURRENT_TIMESTAMP)";
$result = mysqli_query($connect,$query);
```

Ceux-ci ne sont pas très complexes. On récupère les informations de l'url avec $_GET["Nom"] Puis, il suffit de créer une requête SQL, et de l'exécuter. Dans l'exemple ci-dessus, il s'agit d'une requête INSERT utilisée pour générer le log de connexion des cartes, mais on peut également utiliser UPTADE.

```
$Num= $_GET["Num"];
$Acces = $_GET["Acces"];
$query = "UPDATE `acces_badge` SET `Numero`='$Num',`Type_acces`='$Acces' WHERE `Numero` = '$Num' ";
$result = mysqli_query($connect,$query);
```

Celle-ci est utilisée pour changer les permissions des cartes.

3.6 Fichiers de collecte

Ceux-ci sont plus longs car il faut, d'une part, récupérer les informations nécessaires mais également les renvoyer à l'esp ou à l'application Android.

```
$idList = "";
$query = "SELECT * FROM `acces_badge`";
$result = mysqli_query($connect,$query);
if($result -> num_rows > 0)
{
        while($row = $result -> fetch_assoc())
        {
            $idList = $idList . strval($row["Type_acces"]) . "/";

        }
}
$result = $idList;
echo $result;
echo "9999/#";
```

Comme pour les fichiers d'envoi, on crée une requête SQL qu'on exécute mais ici on stocke le résultat (ce que le dB nous envoie) dans une variable. Mais ce que la db nous envoie est un objet et est donc inutilisable en l'état.

Pour extraire les données, on parcourt l'objet avec une boucle, on récupère le contenu de chaque ligne qu'on stocke dans une variable. Pour faciliter la récupération par l'ESP et l'application Android, on rajoute entre chaque valeur récupérée un caractère de séparation "/".

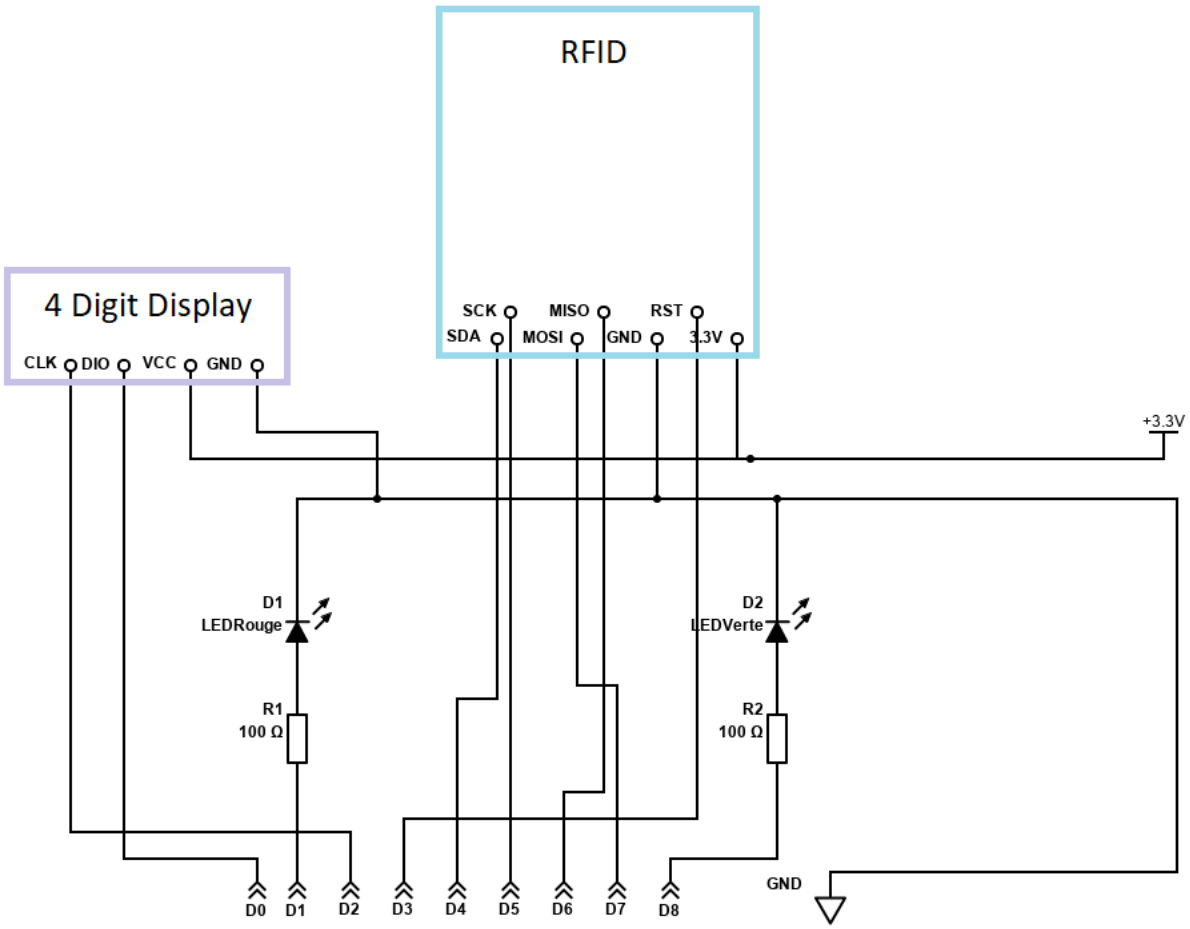
Quand tout cela est fini, on affiche le contenu de la variable sur la page. En effet l'ESP et l'application Android vont récupérer le contenu de cette page y compris le résultat de variables leur permettant ainsi d'avoir les données dont ils ont besoin.

4 Système embarqué

4.1 Introduction

Cette partie du rapport se focalisera sur la partie système embarqué, du schéma électronique en passant par l'ordinogramme du programme ainsi que quelques explications sur les parties les plus importantes de ce dernier. La partie système embarqué est centrale dans ce projet, car elle permet de pouvoir lire les carte RFID mais aussi de déterminer leur autorisation. Sans elle impossible de pouvoir utiliser le projet ou même d'utiliser les autres parties de celui-ci .

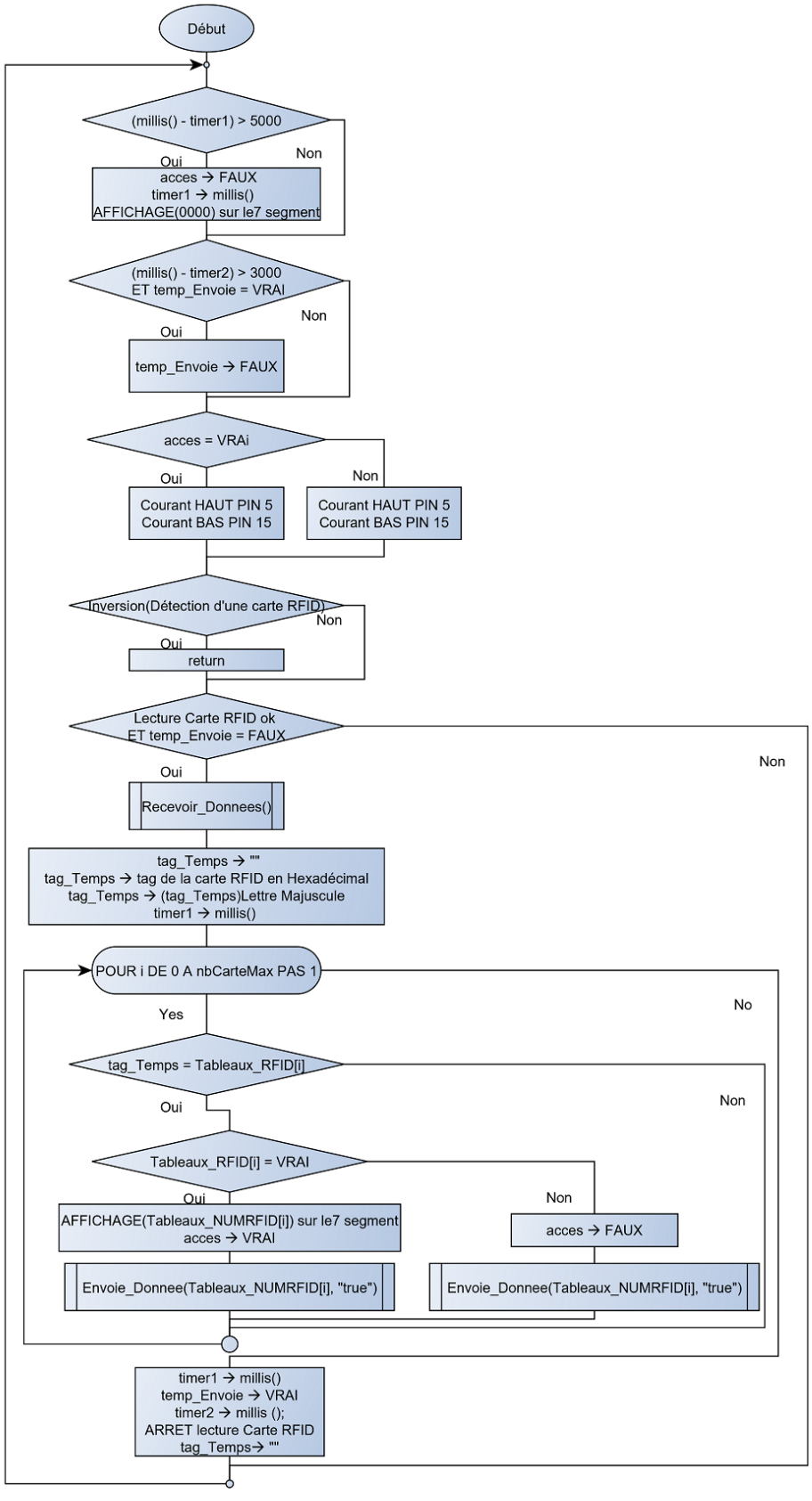
4.2 Schéma électronique



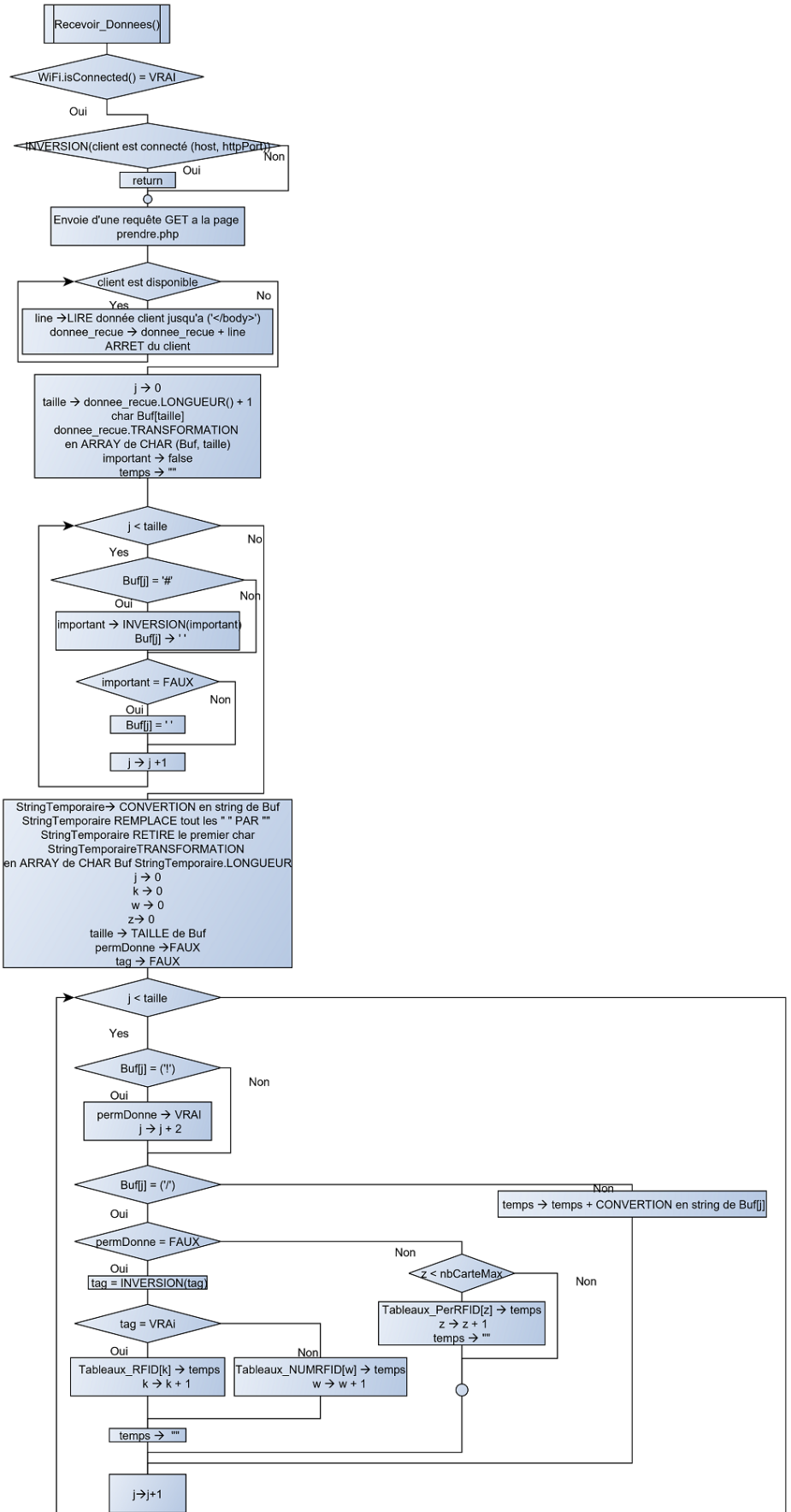
Ceci est le schéma électronique du projet. On peut voir qu' on a utilisé la grande partie des ports disponibles de l'esp. De plus tous les branchements sont situés du même côté de l'esp.

4.3 Ordinogramme

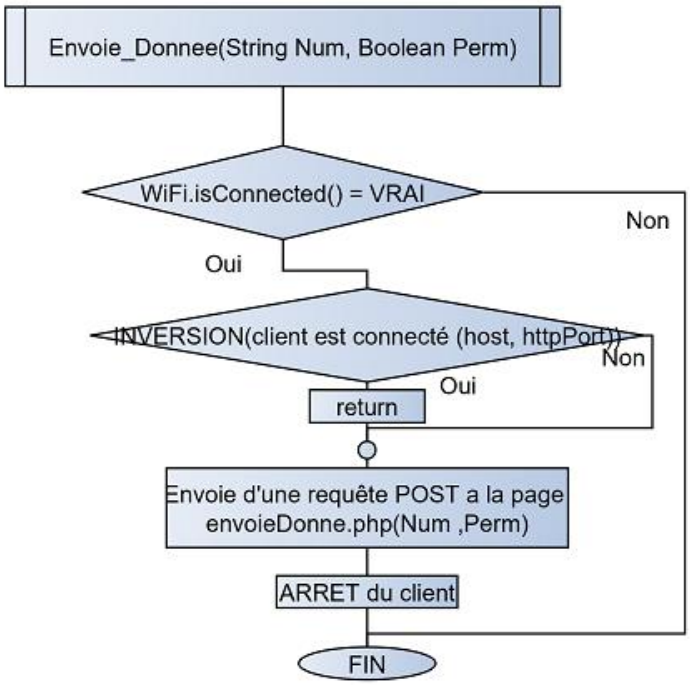
Celui-ci est divisé en 3 différentes parties. D'abord la fonction Loop de l'esp. Il s'agit du programme principal. Celui-ci s'occupe de la vérification des cartes, de l'allumage des leds et la gestion du 7 segments.



Ensuite vient la plus grande fonction du programme : la fonction Recevoir_Donnees(). Celle-ci permet d'aller chercher des données dans la db, de les trier correctement et de les stocker dans les différents tableaux de l'esp .



Pour finir, vient la première fonction Envoie Donnee. Celle-ci est très simple et sert à envoyer les logs de connexion à la dB. Elle accepte 2 arguments : le numéro de la carte ainsi que la permission de la carte.



5 Explication du code de l'esp

5.1 Timer et leds

```
if ((millis() - timer1) > 5000)
{
  display.showNumberDec(0000);
  acces = false;
  timer1 = millis();
}
if ((millis() - timer2) > 3000 && temp_Envoie)
{
  temp_Envoie = false;
}
//lampe
if (acces)
{
  digitalWrite(15, HIGH);
  digitalWrite(5, LOW);
}
else
{
  digitalWrite(5, HIGH);
  digitalWrite(15, LOW);
}
```

Cette partie gère 2 choses principalement, 2 timer et les leds. Les timer : le premier permet de refermer l'accès après 5 secondes, si une carte avec une autorisation valide a été lue. Le second permet de ne pas lire de carte si une a déjà été lue récemment (3 secondes entre chaque lecture de carte).

Les leds fonctionnent avec une variable booléenne, si celle-ci est à l'état true alors la led verte est allumée, si elle est à l'état false alors la rouge est allumée.

Si une des 2 leds est allumée l'autre est alors forcément éteinte.

5.2 Lecture et vérification de carte

```
if (rfid.readCardSerial() && temp_Envoie == false)
{
  Recevoir_Donnees();
  tag_Temps = "";
  tag_Temps = String(rfid.serNum[0], HEX) + String(rfid.serNum[1], HEX) + String(rfid.serNum[2],
  HEX) + String(rfid.serNum[3], HEX) + String(rfid.serNum[4], HEX);
  tag_Temps.toUpperCase();
  timer1 = millis();
```

Cette partie gère la vérification de l'ID des cartes RFID. Si une carte est détectée et que son contenu est lisible (première condition), alors on commence par aller chercher les données des cartes dans la db (au cas où il y aurait eu un changement) grâce à la fonction Recevoir_Donnees(). Puis on lit le contenu de la carte en Hexadécimal, et on le met en majuscules (format utilisé dans la base de données).

```
for (int i = 0; i < nbCarteMax; i++)
{
  if (tag_Temps == Tableaux_RFID[i])
  {
    if (Tableaux_PerRFID[i] == "true")
    {
      Serial.println("Acces OK");
      display.showNumberDec((Tableaux_NUMRFID[i]).toInt());
      acces = true;
      Envoie_Donnee(Tableaux_NUMRFID[i], "true");
    }
    else//acces KO
    {
      Serial.println("Acces refusé");
      acces = false;
      Envoie_Donnee(Tableaux_NUMRFID[i], "false");
    }
  }
}
```

Cette boucle permet de parcourir un tableau dans lequel sont stockés les tags des cartes. Une fois le bon tag trouvé, on regarde si celui-ci a un accès, si oui on autorise l'accès et affiche son numéro sur le 7 segments. Sinon la led rouge reste éteinte. Dans tous les cas on envoie un log à la dB avec les fonctions Envoie_Donnee();

5.3 Envoi de données à la DB

```
void Envoie_Donnee(String Num, String Perm)
{
  if (WiFi.isConnected())
  {
    if (!client.connect(host, httpPort)) {
      return;
    }
    client.print(String("POST ") + "/envoieDonne.php?&Mdp=" + (mdp) + "&Num=" + (Num) + "&Acces=" + (Perm) + " HTTP/1.1\r\n"
     + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n");
    client.stop();
  }
}
```

Cette fonction permet d'envoyer des données à la db grâce à une méthode POST.

En premier lieu, on vérifie que l'esp est bien connecté au wifi, ensuite on se connecte à une page PHP, celle-ci fait la liaison entre l'esp et la base de données. Pour les logs, on ne doit envoyer que 3 informations : un mot de passe, le tag de la carte, ainsi que, si elle a un accès ou pas, le moment de la connexion est géré par la db. Le mot de passe permet d'utiliser le fichier PHP. Il s'agit d'une protection. Pour cela on doit intégrer ces informations dans l'url. Quand la connexion est finie on stoppe le client.

5.4 Récupération des données de la DB

```
if (WiFi.isConnected())
{
  if (!client.connect(host, httpPort)) {
    return;
  }
  client.print(String("GET ") + "/prendre.php?&Mdp=" + (mdp) + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n");
  client.stop();
  while (client.available()) {
    String line = client.readStringUntil('</body>');
    donnee_recue = donnee_recue + line;
    client.stop();
  }
```

Cette fonction permet de recevoir les données de la dB, grâce à un fichier PHP. Une fois qu'on a vérifié la communication au wifi, et si le site qu'on voulait atteindre était bien atteignable, on récupère tout le contenu de la page (si le mot de passe qu'on a envoyé est le bon).

Sinon on ne récupère aucune information, dont les données, mais également beaucoup d'autres choses inutiles (balise HTML, infos diverses).

5.5 Nettoyage des données Partie 1

```
int j = 0;
int taille = donnee_recue.length() + 1;
char Buf[taille];
donnee_recue.toCharArray(Buf, taille);
bool important = false;
String temps = "";
while (j < taille)
{
  if (Buf[j] == '#')
  {
    important = !important;
    Buf[j] = ' ';
  }
  if ( important == false)
  {
    Buf[j] = ' ';
  }
  j++;
}
```

Cette partie de code permet de faire une bonne partie de nettoyage. En effet, les informations importantes qu'il faut récupérer sont situées entre 2 caractères spéciaux :"#". On transforme notre string en char et on le parcourt. On remplace tout caractère qu' on trouve par un espace (on ne peut pas le remplacer par rien). Mais si on rencontre un "#" alors on garde cette partie jusqu'au prochain "#", on efface également les "#" dans le processus.

5.6 Nettoyage des données Partie 2

```
String StringTemporaire = String(Buf);
StringTemporaire.replace(" ", "");
StringTemporaire.remove(0, 1);
StringTemporaire.toCharArray(Buf, StringTemporaire.length());
j = 0;
int k = 0;
int w = 0;
int z = 0;
taille = sizeof(Buf);
bool permDonne = false;
bool tag = false;
```

Cette partie permet de supprimer tout l'espace créé par la partie du dessus. Ainsi qu'initialiser un grand nombre de variables pour la prochaine partie.

5.7 Nettoyage des données Partie 3

Cette partie va extraire les informations, de base dans notre texte, elles sont réparties comme cela :

/TAG/NUM/……/TAG/NUM/!/false/true /false……/9999

La *ligne* est divisée en 2 parties. La première contient les tags et les numéros des cartes. La seconde, quant à elle, contient les permissions. A la fin de la ligne, on a un /9999. Son utilité est d'éviter un bug. En effet, si jamais on retire le 9999, la partie après le / sera totalement vide, l'esp va tenter de lire ces données vides et alors la dernière permission sera remplie de caractères parasites.

```
while (j < taille)
{
  if ((Buf[j] == ('!')))
  {
    permDonne = true;
    j = j + 2;
  }
  if ((Buf[j] == ('/')))
  {
    if (permDonne == false)
    {
      tag = !tag;
      if (tag == true)
      {
        Tableaux_RFID[k] = temps;
        k = k + 1;
      } else
      {
        Tableaux_NUMRFID[w] = temps;
        w = w + 1;
      }
      temps = "";
    }
    else
    {
      if (z < nbCarteMax) {
        Tableaux_PerRFID[z] = temps;
        z = z + 1;
        temps = "";
      }
    }
  }
  else
  {
    temps = temps + String(Buf[j]);
  }
  j++;
}
```

La boucle for permet de parcourir un tableau de char. On stocke les caractères un à un dans une variable temporaire. Quand on rencontre le caractère "/" , alors on met le contenu de la variable dans l'un de nos tableaux de données (ils contiennent les tags, les numéros de tag et leur permission). Quand on rencontre le caractère "!" , on change juste de tableau.

5.8 Notes

Toute cette partie aurait pu être plus rapide car il existe en C une fonction nommée strtok().

Celle-ci permet de séparer les différents éléments d'un tableau de char avec un élément séparateur.

Mais celle-ci s'est avérée problématique car elle a tendance à écrire des données à des emplacements mémoire déjà utilisés par d'autres variables, comme un tableau de données, ou alors le buffer de lecture normalement utilisé par la carte RFID, sans parler des nombreux crashs provoqués par la fonction quand celle-ci veut écrire à des emplacements mémoire du logiciel présents sur la carte. Pour éviter d'endommager le logiciel de la carte j'ai alors décidé d'arrêter de l'utiliser. Et j'ai donc créé le code juste au-dessus, moins compact que la fonction strtok() mais plus sûr.

```
18:22:22.378 -> Trié
18:22:22.378 -> tag:                 ?□□ @ NumRFID : Permission :
tag: 939A7197EF NumRFID :01 Permission : true
18:22:22.470 -> tag: D3F17997CC NumRFID :02 Permission : true
18:22:22.470 -> tag: 63F873977F NumRFID :03 Permission : true
18:22:22.563 -> tag: F3A4949754 NumRFID :04 Permission : false
18:22:22.609 -> tag: B3188797BB NumRFID :05 Permission : false
18:22:22.655 -> tag: 13B07E974A NumRFID :06 Permission : false
18:22:22.701 -> tag: 93F4F7966 NumRFID :07 Permission : false
18:22:22.747 -> tag: 5324699789 NumRFID :08 Permission : false
18:22:22.793 -> tag: D37A75974B NumRFID :09 Permission : false
18:22:22.839 -> tag: C387569785 NumRFID :10 Permission : false
18:22:22.885 -> tag: E383997FE NumRFID :11 Permission : false
18:22:22.978 -> tag: 638D5E9727 NumRFID :12 Permission : false
```

On peut voir ici que le premier élément d'un des tableaux a été remplacé par des caractères inconnus.

6 *English project*

This part is the project part, it will be a small resume of the principal aspect of the project and why I made a certain choice.

To start let's talk about the esp8266, which is the embedded system part of the project. It was a simple part accepted for 1 or 2 things, is goal is just to read a card, get the access right from the

dB, send a log to the dB and open it if the card has access. Each time a card gets scan, the esp8266 go get the access right from the dB, I made things like this because I want to be sure to have the right data because in the early day the program was working differently. The program was taking data from the dB every 10 seconds, which was working great, but it causes an issue: if I modify the access, and a card is scan less than 5 sec after, the esp will check an old version of the access.

The middleware is a sort of bridge between the esp./ android app and the database.

The choice of using PHP, was because it was fast and easy to do, and it works well. But it causes issues later especially with the esp8266 because it doesn't have a split function and when I get the data using HTTP requests, I get a bunch of useless stuff. The Android app was easy because split exists.

For hosting the middleware in the first place I wanted to use Xamp, it this like wamp but more recent, and with more options. Unfortunately, it never worked properly on my pc. After some time, I finally choose Wamp, and this work fine. Wamp can still do a lot of things, I can host a database MySQL, and it can host webpage or PHP files using Apache.

Now for the android part is goal was to allow the user to see logs and modify the database. I don't have much to say, everything in my application had been seen during the lesson. But despite that, that was quite difficult for me because in the first place I'm not good with design or UI in general. So, I take a simple solution, a blue background a white background for the text or button. It works, this is not the best-looking application, but it works fine. In the second place, the recycler views were hard to implement.it is the thing that allows me to display my data on the log and access right windows, and to work that as take some time.

The last part of the project was the box, the first design of it was costly and useless. Because my first idea was to do a 100% metallic box, this is bad for numerous reasons, it would be costly, not easy to do because I don’t have the material to cut through metal, finally because the esp is using wifi, and metal don't let radio wave pass through it.

So, in the end, I use Makerbeam for the structure and white foam for the wall. The white foam is more difficult to cut than I was thinking, and there are 2 plates per wall, so it took a very long time to cut all the walls. But the assemblage was fast

553 words

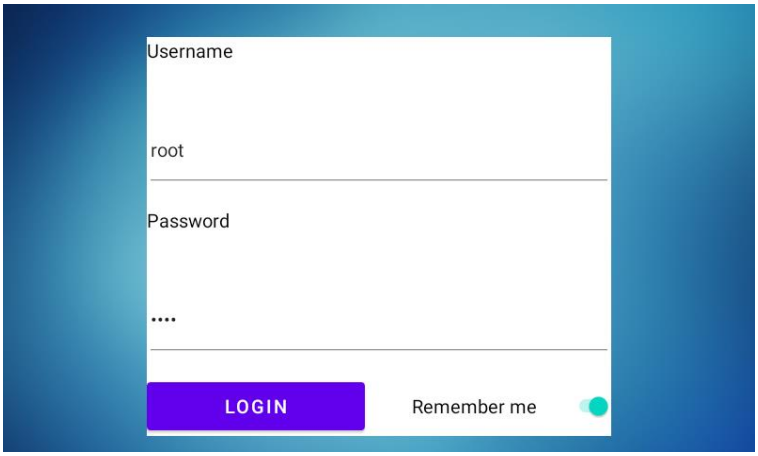
7 *Développement mobile*

7.1 Introduction

Cette partie du rapport se focalisera sur l'application Android, de son fonctionnement global à la récupération et modification de données dans la base de données, ainsi qu'à l'affichage de données.

Cette partie n'est pas aussi importante que les 2 précédentes, en effet le système peut fonctionner sans problème sans lui. Mais il sera alors verrouillé, et il sera impossible de modifier les permissions (sans passer par la base de données directement), et de voir les logs de connexion.

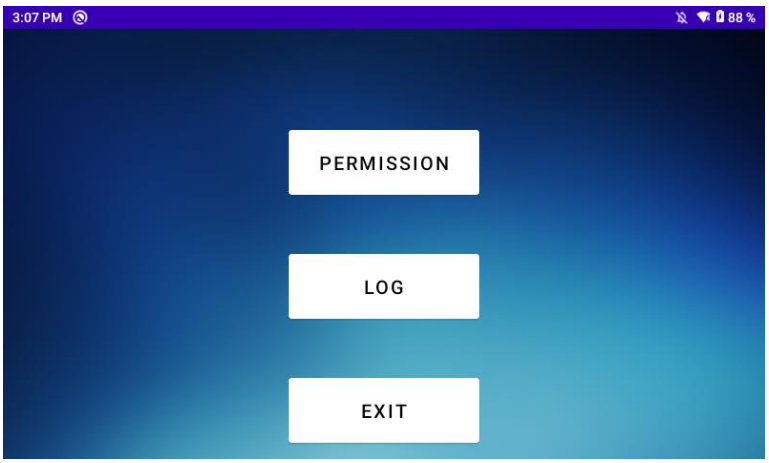
7.2 Fonctionnement général de l'application



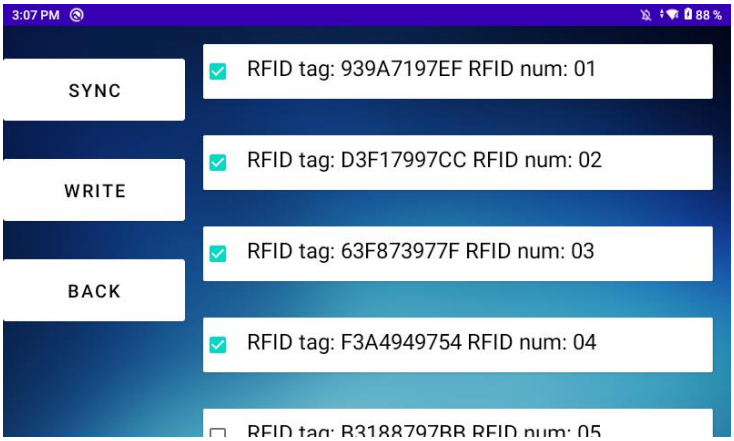
Le fonctionnement de l'application est assez simple. Quand on la démarre, on arrive sur une page de login où il faut rentrer un identifiant et un mot de passe pour pouvoir accéder à la suite.

Il y a également une option pour pouvoir sauvegarder le mdp et l'identifiant si on coupe l'application, idéal pour ne pas devoir rentrer le mot de passe à chaque fois.

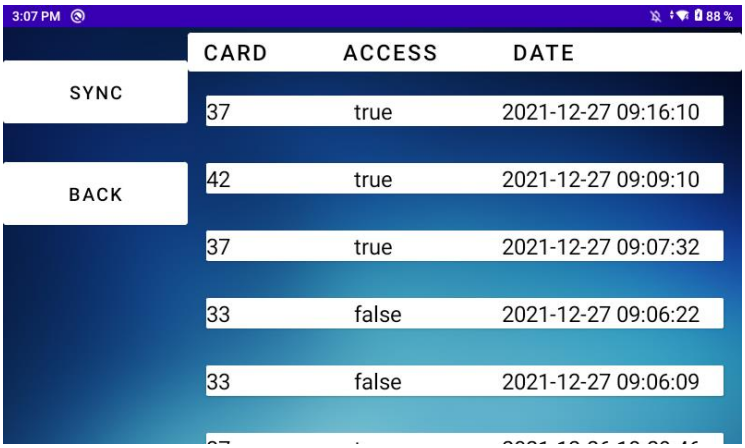
Néanmoins, si cette option n'est pas active alors l'application va effacer le mdp et l'identifiant enregistrés.



La seconde page est une simple page de menu avec 3 boutons : un bouton pour revenir à la page précédente et 2 boutons pour se rendre soit dans la page d'autorisation soit dans celle des logs.



La page autorisation est composée de 2 parties. A droite la liste des cartes RFID, permettant de les parcourir et de changer les permissions d'accès des cartes. A gauche 3 boutons : le premier sert à synchroniser la liste des cartes avec la base de données, le second permet d'écrire les modifications apportées dans la liste à la base de données et le dernier pour revenir à la page de menu.



La dernière page de log est, elle aussi, composée de 2 parties. A gauche un bouton de synchronisation et de retour à la page menu, et à droite : la liste des logs de connexion des cartes .

8 Explication du code

```
PATH = filesDir.absolutePath
var fichier : File = File( pathname: "$PATH/$filename")

if(fichier.exists()){
    val fr = BufferedReader(FileReader(fichier))
    var ligne = fr.readLine()
    ident= ligne.split( ...delimiters: '\t')[0]
    mdp= ligne.split( ...delimiters: '\t')[1]
    Identifiant?.setText(ident)
    MDP?.setText(mdp)
    fr.close()
}
```

Page du Login

Cette partie de code s'exécute au lancement de l'application. Elle vérifie si un fichier de sauvegarde est présent. Si oui, on lit le fichier et on préremplit les zones que l'utilisateur utilise pour se connecter. Si aucun fichier n'est présent alors l'utilisateur doit tout rentrer à la main.

```
if(Souvenir?.isChecked == true)
{
    ident = (Identifiant?.text).toString()
    mdp = (MDP?.text).toString()
    var fw = FileWriter(fichier)
    fw.write( str: "")
    fw.write( str: "$ident\t$mdp")
    fw.flush()//sync
    fw.close()
}else {
    var fw = FileWriter(fichier)
    fichier.delete()
    fw.close()
}
```

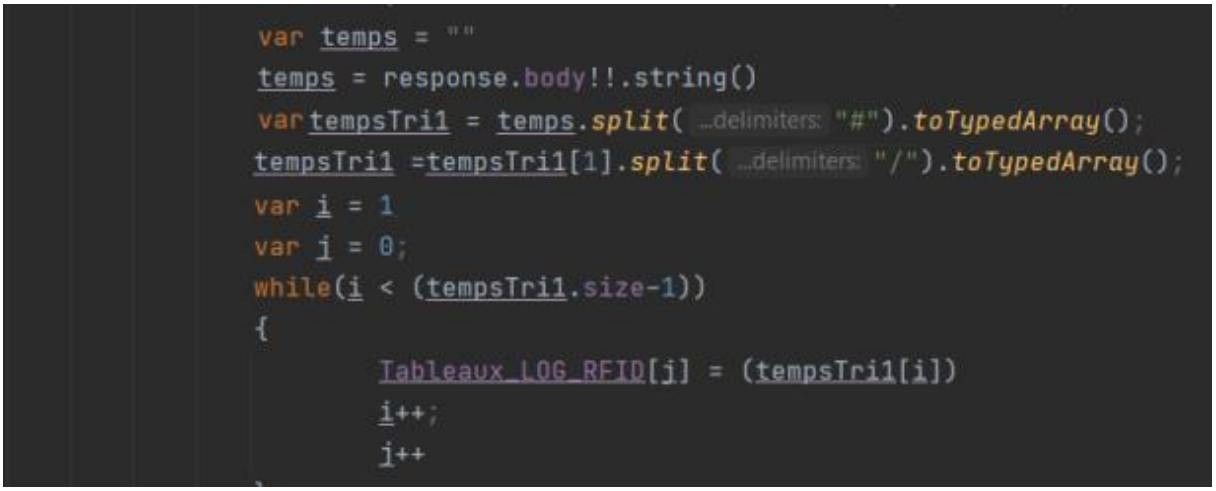
On a le choix de pouvoir sauvegarder son mot de passe ainsi que son identifiant. Cette partie permet de les sauvegarder dans un fichier interne de la tablette. Si le fichier n'est pas présent alors on le crée automatiquement. Cependant, si jamais on ne veut pas sauvegarder son mdp et identifiant, alors on supprime le fichier contenant la sauvegarde.

Si jamais le mdp ou l'identifiant ne sont pas les bons, un petit TOAST va prévenir l'utilisateur, mais sans lui informer lequel est incorrect, rendant l'utilisation du brut force sur l'application plus compliqué.

8.1 Page des Logs

```
fun RecuperationD()
{
    var url = "http://192.168.137.1/prendreLog.php?&Mdp=$mdp"
    val request = Request.Builder()
        .url(url)
        .build()
    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
        }
        override fun onResponse(call: Call, response: Response) {
            response.use { it: Response
                if (!response.isSuccessful) throw IOException("Unexpected code $response")
```

Comme pour l'esp, on utilise un fichier PHP pour pouvoir aller prendre les informations de la base de données grâce à une fonction nommée Récupération. Celle-ci est exécutée à l'appui du bouton SYNC. Pour cela on crée une requête http et on attend sa réponse. Celle-ci contenant toutes les informations dont nous avons besoin. Comme pour l'esp, pour que le PHP marche et nous envoie des informations, on doit mettre un mot de passe dans l'url.



```
var temps = ""
temps = response.body!!.string()
var tempsTri1 = temps.split( ...delimiters: "#").toTypedArray();
tempsTri1 =tempsTri1[1].split( ...delimiters: "/").toTypedArray();
var i = 1
var j = 0;
while(i < (tempsTri1.size-1))
{
        Tableaux_LOG_RFID[j] = (tempsTri1[i])
        i++;
        j++
```

Comme pour l'esp, les informations utiles sont séparées par "#" puis par un "/" entre chacune d'elles. Ici, l'utilisation d'un simple split nous permet de tout séparer et de tout récupérer .

Les informations sont ensuite stockées dans un Array et affichées sur la page. Pour l'affichage, on utilise une recyler views, celle-ci est optimisée pour pouvoir parcourir un grand nombre d'éléments. Elle contient une simple ligne de texte et son contenu est notre information, ici, le numéro de la carte, son accès au moment du scan et l'heure du scan.

8.2 Page des permissions

Comme pour la page des Logs, cette page va également chercher des informations dans la dB. La méthode étant très similaire, elle ne sera pas réexpliquée ici. La seule différence majeure est qu'il y a 3 Arrays à remplir ici, les numéros, les tags et les permissions, par conséquent la récupération est un peu plus complexe. Cependant elle repose également sur l'utilisation des split. La récupération des données s'effectue également avec l'appui du bouton SYNC.

```
fun Ecrire(v : View)
{
    var Tempsischecked = Array<Boolean>( size: 27){false}
    Tempsischecked = adapter?.ischecked!!
    for(i in 0..26)
    {
        if(!(tableaux_perRFIDBACKUP[i] == Tempsischecked[i].toString()))
        {
            tableaux_perRFIDBACKUP[i] = Tempsischecked[i].toString()
            UptadeD(Tempsischecked[i].toString(),Tableaux_NUM_RFID[i])
```

Avant l'envoi des modifications sur les accès des cartes, on regarde d'abord quelle carte a eu son accès changé. En effet, il ne sert à rien de modifier la base de données alors que la valeur n'a pas bougé. Cette opération prend de la mémoire et de la bande passante pour rien. Pour vérifier cela, on regarde l'état des permissions actuelles avec celle prise juste après être allé chercher les données dans la db. S' il y a une différence alors on change l'accès.

```
fun UptadeD(Perm : String,Num : String)
{
    var uRL = ""
    val request = Request.Builder()
        .url( url: "http://192.168.137.1/modifDonne?&Mdp=$mdp&Num=$Num&Acces=$Perm")
        .build()
    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            e.printStackTrace()
        }
```

La fonction UptadeD modifie donc le contenu de la dB, comme pour le reste nous envoyons une requête à un fichier PHP. Celle-ci contient le numéro de la carte et son accès. L'accès pour changer celui-ci et le numéro de la carte pour pouvoir la retrouver dans la base de données.

9 Création de la boite

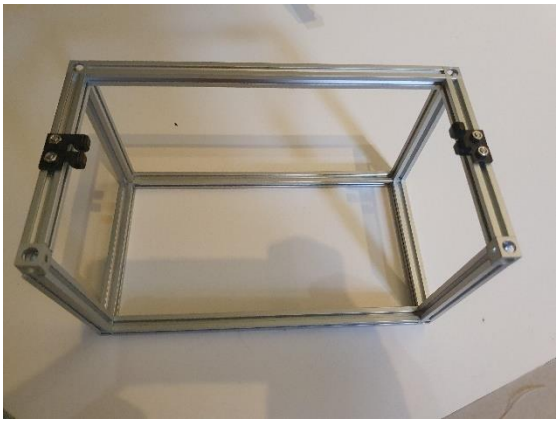
9.1 Introduction

La boite est très importante. Elle va, d'une part, contenir la partie électronique du projet, et d'autre part, elle reflètera la qualité du projet.

9.2 Création de l'armature

L'armature de la boite est assez simple. Il s'agit d'un parallélépipède rectangle fait entièrement en MakerBeam. Ceux-ci sont des barres d'aluminium, qu'on peut facilement relier entre elles grâce à des boulons, écrous et d'autres type d'attache.

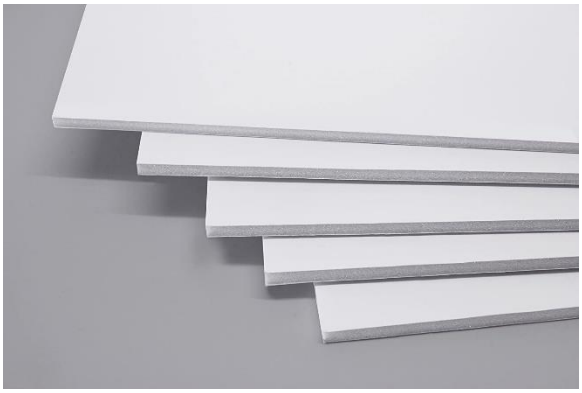
Au-dessus du parallélépipède rectangle, se trouvent 2 petits rectangles, ceux-ci sont attachés à des charnières et permettent d'ouvrir le toit de la boite pour pouvoir regarder à l'intérieur. Pour relier les pièces de l'armature principale entre elles, on a utilisé des cubes, ceux-ci, en plus d'augmenter légèrement la taille du cube, sont beaucoup plus beaux et améliorent l'esthétique de la boite.



Malgré l'armature en métal, celle-ci est suffisamment espacée pour permettre à l'esp de pouvoir communiquer en wifi. Un des premiers designs de la boite impliquait de la faire entièrement en métal mais cette idée a vite été abandonnée dû à la très grande possibilité que cela aurait empêché toute communication wifi avec la base de données.

9.3 Installation des panneaux

L'armature étant finie, il a fallu combler le vide entre les barres de l'armature. Pour cela on a utilisé des plaques de mousse blanche, celles-ci sont bon marché et très simples à découper. Les plaques sont mises dans la glissière des barres et ne sont donc maintenues en place que par la force de l'armature sur celles-ci. Chaque "mur" de la boite est constitué de 2 panneaux de mousse, le fait d'en mettre 2 a plusieurs avantages. Le "mur" est beaucoup plus solide qu'avec 1 seul, mais encore cela empêche le "mur" de se déplacer légèrement (réduire l'espace entre la plaque et les barres d'armature), et maintient donc le tout bien en place. Mais l'inconvénient est qu'il faut découper 2 fois plus de plaques.



Les plaques ne sont pas des simples carrés car les cubes n'ont pas de glissières comme les barres. Il a fallu découper les plaques d'une manière bien précise pour que tout cela fonctionne.

Les coins ont été retirés sinon ils allaient bloquer les cubes.

Il a fallu faire des trous dans les plaques pour permettre l'affichage du 7 segments ou encore la vue des leds. Un autre trou a été fait sur une plaque du côté de la boite pour permettre la sortie du câble USB. Il n'y a pas de trous pour le lecteur RFID car celui-ci arrive sans problème à lire les cartes au travers des plaques.

9.4 L'intérieur de la boite

La partie électronique est fixée à la boite par du papier collant sauf pour la breadboard qui est fixée avec son propre auto-collant et les leds qui sont, elles, enfoncées fermement dans des trous. Malgré le papier collant, les fixations sont assez solides. Les câbles sont également liés entre eux grâce à du papier collant pour améliorer la clarté du câblage.

10 Conclusion

La réalisation du projet s'est passée dans cette ordre : ESP8266, la base de données et middleware et enfin le programme Android.

La partie de L'esp8266 était dans l'ensemble assez simple ainsi que la lecture de la carte et la gestion des droits d'accès également. Une fois cette partie finie, j'ai fait la base de données, également assez simple et ensuite je suis passé au middleware.

Cette partie fut beaucoup plus compliquée, j'ai tenté des approches différentes mais celle qui a le mieux marché fut finalement l'utilisation de PHP. J'ai dû, après cela, modifier le code de l'ESP8266. Ça a pris beaucoup de temps, notamment dû à la fonction strtok() qui ne marchait pas bien.

Une fois tout cela fini, je suis enfin passé à la partie Android, celle-là était beaucoup plus longue et assez dure, mais j'ai finalement réussi à faire quelque chose de beau et de fonctionnel.

Je suis juste déçu de ne pas avoir eu le temps de rajouter des fonctionnalités ou d'améliorer mon application Androïde pour la rendre plus jolie. Mais également, j'avais des idées supplémentaires pour la boite. Le manque de temps et le faible nombre de ports disponibles sur l'esp8266 m'ont fait abandonner ces idées.

Pour conclure, je dirais que le projet exploite assez bien les différents programmes / matériels qu'il utilise et qu'il est une bonne évaluation pour la vérification de nombreuses compétences.

Le choix du middleware est très bon également, il nous force à chercher et tenter différentes approches.

Cette expérience fut enrichissante et nous prépare bien au stage à venir .

11 Médiagraphie

MakerBeam: https://www.makerbeam.com/

WampServer: https://www.team-ever.com/installer-et-parametrer-wamp-sur-windows/

12 Lexique

Middleware : logiciel permettant l'échange d'informations entre 2 applications / logiciels différents

PHP: langage de programmation orienté objet utilisé pour la création de pages web

Arduino: microcontrôleur programmable de marque Arduino

ESP8266/ESP: microcontrôleur programmable de marque Espressif Systems

RFID: radio identification, méthode de récupération et de modification d'informations à distance, utilisant des radio-étiquettes (ici les cartes RFID).

13 Annexe

13.1 Variables ESP8266

```
const int CLK = 4

const int DIO = 16

TM1637Display display(CLK, DIO)

#define SS_PIN 2

#define RST_PIN 0

RFID rfid(SS_PIN, RST_PIN)

const char* ssid     = "PC1"

const char* password = "11111111"

const char* host = "192.168.137.1"

String donnee_recue = ""

bool syncro = true

Pinger pinger

WiFiClient client
```

```
String tag_Temps = ""
int const nbCarteMax = 27
String Tableaux_RFID[nbCarteMax]
String Tableaux_PerRFID[nbCarteMax]
String Tableaux_NUMRFID[nbCarteMax]
bool temp_Envoie = false
bool acces = false
double timer1 = 0
double timer2 = 0
bool lecture = false
const int httpPort = 80
String mdp = "EZqVtcX4571"
```

13.2 Code Esp8266

```
#include <SPI.h>
#include <RFID.h>
#include <TM1637Display.h>

#include <Pinger.h>
#include <PingerResponse.h>
#include <ESP8266WiFi.h>


const int CLK = 4;
const int DIO = 16;
TM1637Display display(CLK, DIO);

#define SS_PIN 2
```

```
#define RST_PIN 0

RFID rfid(SS_PIN, RST_PIN);


const char* ssid     = "PC1";
const char* password = "11111111";
const char* host = "192.168.137.1";
String donnee_recue = "";
bool syncro = true;
Pinger pinger;
WiFiClient client;


String tag_Temps = "";

int const nbCarteMax = 27;

String Tableaux_RFID[nbCarteMax];
String Tableaux_PerRFID[nbCarteMax];
String Tableaux_NUMRFID[nbCarteMax];

bool temp_Envoie = false;

bool acces = false;
double timer1 = 0;
double timer2 = 0;
bool lecture = false;
const int httpPort = 80;
```

```
String mdp = "EZqVtcX4571";
void setup() {

  display.setBrightness(0x0a);

  pinMode(15, OUTPUT);
  pinMode(5, OUTPUT);

  SPI.begin();
  rfid.init();

  display.showNumberDec(0000);

  WiFiClient client;
  Serial.begin(9600);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  timer1 = millis();
  timer2 = millis();
}
void loop()
{
  if ((millis() - timer1) > 5000)
  {
    display.showNumberDec(0000);
    acces = false;
    timer1 = millis();
  }
```

```
  if ((millis() - timer2) > 3000 && temp_Envoie)
  {
    temp_Envoie = false;
  }

  if (acces)
  {
    digitalWrite(15, HIGH);
    digitalWrite(5, LOW);
  }
  else
  {
    digitalWrite(5, HIGH);
    digitalWrite(15, LOW);
  }
  if (!rfid.isCard())
  {
    return;
  }
  if (rfid.readCardSerial() && temp_Envoie == false)
  {
    Recevoir_Donnees();
    tag_Temps = "";
    tag_Temps = String(rfid.serNum[0], HEX) + String(rfid.serNum[1], HEX) + String(rfid.serNum[2], HEX) +
String(rfid.serNum[3], HEX) + String(rfid.serNum[4], HEX);
    tag_Temps.toUpperCase();
    Serial.println("              ");
    timer1 = millis();
    for (int i = 0; i < nbCarteMax; i++)
    {
```

```
      if (tag_Temps == Tableaux_RFID[i])
      {
        if (Tableaux_PerRFID[i] == "true")
        {
          Serial.println("Acces OK");
          display.showNumberDec((Tableaux_NUMRFID[i]).toInt());
          acces = true;
          Envoie_Donnee(Tableaux_NUMRFID[i], "true");
        }
        else//acces KO
        {
          Serial.println("Acces refusé");
          acces = false;
          Envoie_Donnee(Tableaux_NUMRFID[i], "false");
        }
      }
    }
    temp_Envoie = true;
    timer1 = millis();
    timer2 = millis ();
    //clear
    rfid.halt();
    tag_Temps = "";
  }
}
void Envoie_Donnee(String Num, String Perm)
{
  if (WiFi.isConnected())
  {
```

```
    if (!client.connect(host, httpPort)) {
      return;
    }
    client.print(String("POST ") + "/envoieDonne.php?&Mdp=" + (mdp) + "&Num=" + (Num) + "&Acces=" + (Perm) + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n");
    client.stop();
  }
}
void Recevoir_Donnees()
{
  if (WiFi.isConnected())
  {
    if (!client.connect(host, httpPort)) {
      return;}
    client.print(String("GET ") + "/prendre.php?&Mdp=" + (mdp) + " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: keep-alive\r\n\r\n");
    client.stop();
    while (client.available()) {
      String line = client.readStringUntil('</body>');
      donnee_recue = donnee_recue + line;
      client.stop();
    }
    int j = 0;
    int taille = donnee_recue.length() + 1;
    char Buf[taille];
    donnee_recue.toCharArray(Buf, taille);
    bool important = false;
    String temps = "";
    while (j < taille)
    {
```

```
  if (Buf[j] == '#')
  {
    important = !important;
    Buf[j] = ' ';
  }
  if ( important == false)
  {
    Buf[j] = ' ';
  }
  j++;
}
//on retire les espaces
String StringTemporaire = String(Buf);
StringTemporaire.replace(" ", "");
//retire le premier
StringTemporaire.remove(0, 1);
StringTemporaire.toCharArray(Buf, StringTemporaire.length());
//traitement des données: mise dans un tableau
j = 0;
int k = 0;
int w = 0;
int z = 0;
taille = sizeof(Buf);
bool permDonne = false;
bool tag = false;
while (j < taille)
{
  if ((Buf[j] == ('!')))
  {
```

```
  permDonne = true;
  j = j + 2;
 }
 if ((Buf[j] == ('/')))
 {
  if (permDonne == false)
  {
   tag = !tag;
   if (tag == true)
   {
    Tableaux_RFID[k] = temps;
    k = k + 1;
   } else
   {
    Tableaux_NUMRFID[w] = temps;
    w = w + 1;
   }
   temps = "";
  }
  else
  {
   if (z < nbCarteMax) {
    Tableaux_PerRFID[z] = temps;
    z = z + 1;
    temps = "";
   }
  }
 }
 else
```

```
      {
        temps = temps + String(Buf[j]);
      }
      j++;
    }
    j = 0;
    donnee_recue  = "";
    temps = "";
    client.stop();
  }
}
```