



Dossier technique du projet – Partie personnelle

Etudiant 4

Sommaire

I.	Situation dans le projet.....	1
1.1)	Synoptique de la réalisation.....	1
1.2)	Rappel des tâches de l'étudiant.....	1
1.3)	Contraintes liées au développement.....	2
II.	Conception et mise en œuvre.....	2
2.1)	Fonctionnement du pluviomètre.....	2
2.2)	Réalisation du diagramme de classe.....	3
III.	Envoie des mesures du pluviomètre.....	4
3.1)	Acquisition des mesures.....	4
3.2)	Communication avec la carte Arduino.....	6
IV.	Récupération des données de la base de données.....	7
4.1)	Architecture de la base données.....	7
4.2)	Réalisation des pages PHP.....	7
V.	Mise en place de l'application Android.....	9
5.1)	Architecture de l'application.....	9
5.1.1)	Les différentes activités.....	9
5.1.2)	Les différentes classes.....	13
5.2)	Accès au pages PHP.....	13
5.3)	Décoder le JSON.....	15
VI.	Tests unitaires.....	17
6.1)	Classe CHardware.....	17
6.2)	Classe CLog.....	Erreur ! Signet non défini.
VII.	Conclusion.....	19
VIII.	Glossaire.....	Erreur ! Signet non défini.
IX.	Annexes.....	Erreur ! Signet non défini.

I. Situation dans le projet

1.1) Synoptique de la réalisation



Figure 1 : Les différents systèmes du projet

Au sein du projet j'ai eu pour première tâche d'acquérir les mesures issues d'un pluviomètre depuis une carte Arduino dans le but de pouvoir les communiquer à une carte Arduino pour qu'ensuite elle puisse être enregistrer dans la base de données.

Ma seconde tâche à été de développer une application Android permettant à l'utilisateur (le superviseur de la serre) de pouvoir visualiser en temps réel l'état de fonctionnement de chaque matériel faisant parti du système.

1.2) Rappel des tâches de l'étudiant

Dans ce projet de supervision de serre, j'avais pour premier objectif de mettre en place un pluviomètre permettant de mesurer la quantité de précipitation tombé pendant un intervalle de temps donné. En effet, afin de s'assurer que les plantes aient reçu suffisamment d'eau dans la journée, il est nécessaire de mesurer la pluviométrie.

Mon deuxième objectif à été de mettre en place une application Android permettant à l'utilisateur de pouvoir visualiser en temps réel l'état de fonctionnement de chaque matériel faisant parti du système. A partir de l'application, il est donc nécessaire d'avoir accès à la liste de tous les matériels, que ce soit les capteurs ou les microcontrôleurs. Afin d'assurer un meilleur suivi de l'état de fonctionnement, j'ai décidé d'y intégrer un système d'historique de pannes.

1.3) Contraintes liées au développement

Dans un premier temps, nous avons une **contrainte financière**. Etant donné que le matériel dont j'avais besoin pour mes tâches était mis à disposition dans la section, je n'ai donc pas été touché par cette contrainte.

Ensuite, la **contrainte de développement** m'a fait réaliser l'application Android sous l'IDE Android Studio. Etant donné que j'ai déjà réalisé plusieurs projets scolaires et personnels sous Android Studio, cette contrainte de développement ne m'a pas dérangé.

De plus, nous avons plusieurs **contraintes de qualité**. La première a été que le client final devra pouvoir utiliser le système sans compétence informatique particulière (hormis pour certains paramétrages de configuration). La deuxième contrainte de qualité a été de développer le projet de sorte à pouvoir ajouter facilement de nouvelles mesures par la suite. Par exemple, si le client désire ajouter une mesure pH du sol, le travail à réaliser devra être minimal, voire automatique si possible. Cette contrainte d'évolutivité forte a impliqué une analyse poussée et un travail de développement plus exigeant. La troisième contrainte a été de fournir une documentation complète (dossier de présentation, mode d'emploi, procédure illustrée d'installation) sur le système au client. Nous devons également fournir un exemplaire des sources de nos travaux, ainsi qu'une nomenclature précise du matériel utilisé pour permettre aux bénéficiaires de donner une suite au projet. Une documentation automatique du code devra également être fournie au format html.

Pour terminer, nous avons une **contrainte de fiabilité/sécurité** qui nous a imposé de devoir journaliser en interne tout problème d'acquisition de mesures ou de problème d'accès à la base de données.

Il faut noter que ce projet va être réalisé sur deux années. La première année, où nous sommes, nous nous occupons de la supervision de l'état de la serre, avec récupération et stockage de l'ensemble des données nécessaire. La deuxième année se chargera de l'automatisation de la régulation de la température, l'hydrométrie et de l'intensité lumineuse de la serre.

II. Conception et mise en œuvre

2.1) Fonctionnement du pluviomètre

Tout d'abord, il faut savoir que la quantité d'eau atteignant le sol est exprimée en millimètre. Quand 1 millimètre est dans le pluviomètre cela équivaut à un litre d'eau au mètre carré.

Pour mesurer les précipitations de pluie, le pluviomètre recueille l'eau dans un entonnoir (réceptacle). Si ces précipitations sont sous la forme solide, il est possible d'ajouter des résistances autour du cône pour les faire fusionner. L'eau est ensuite envoyée vers un système contenant un auget basculeur (en rouge sur la Figure 2).

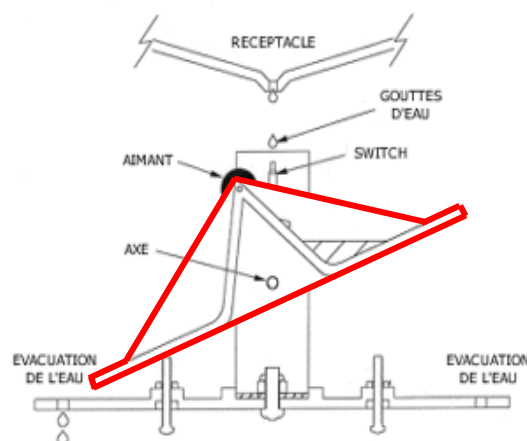


Figure 2 : Fonctionnement d'un pluviomètre

Le pluviomètre contient un switch magnétique qui va permettre d'envoyer une impulsion en sortie afin de pouvoir compter les basculements.



Figure 3 : Etat du switch magnétique au repos

Lorsque l'auget a accumulé 0,2mm d'eau, il bascule. Lors du basculement, l'aimant positionné sur la pointe de l'auget va passer devant le switch magnétique ce qui va fermer le circuit durant un très court instant et donc provoquer une impulsion.

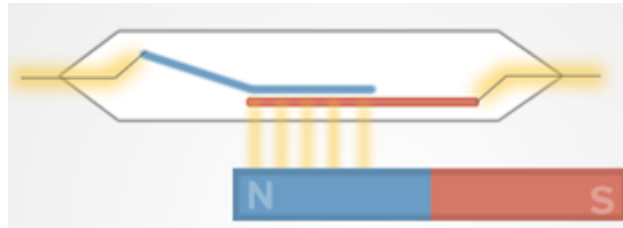


Figure 4 : Etat du switch magnétique lors du basculement

2.2) Réalisation du diagramme de classe

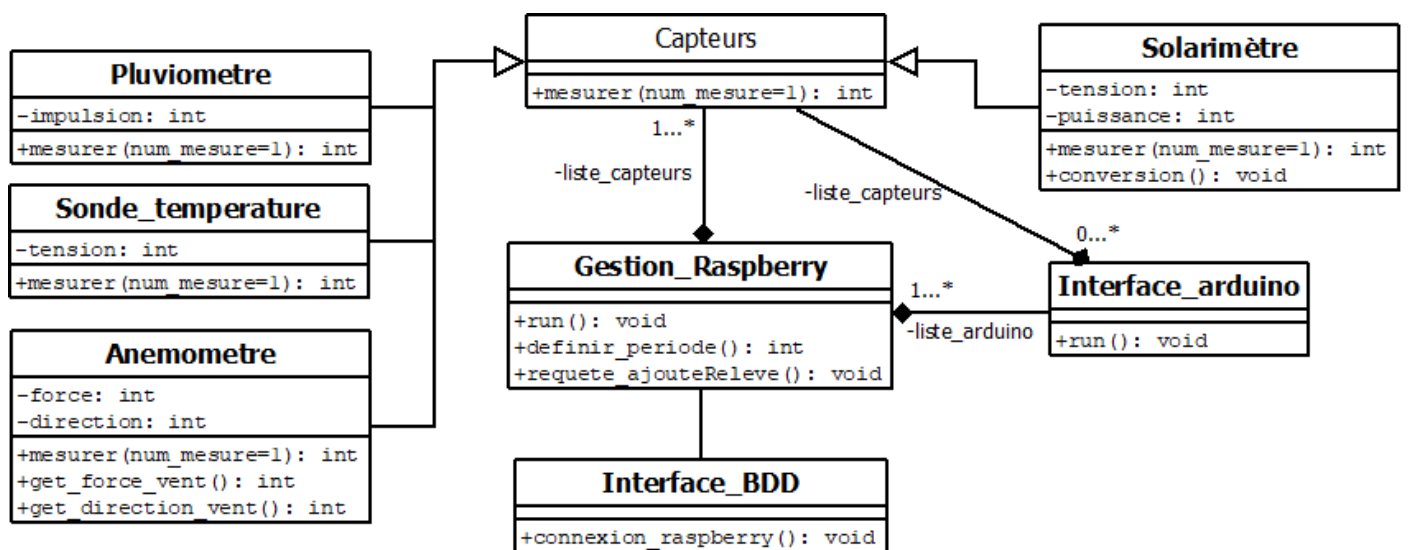


Figure 5 : Diagramme de classe

III. Envoie des mesures du pluviomètre

3.1) Acquisition des mesures

Pour détecter les impulsions émises, j'ai relié le pluviomètre (zone en **bleu** sur la Figure 6 représenté par le symbole d'un switch magnétique) à une carte Arduino (zone en **rouge** sur la Figure 6). Afin de fournir une tension j'ai utilisé une résistance de pull-up (zone en **vert** sur la Figure 6) interne à la carte Arduino. Voici le schéma de câblage représentant les tensions mesurées par la carte Arduino lorsque le switch magnétique est ouvert (pas de basculement).

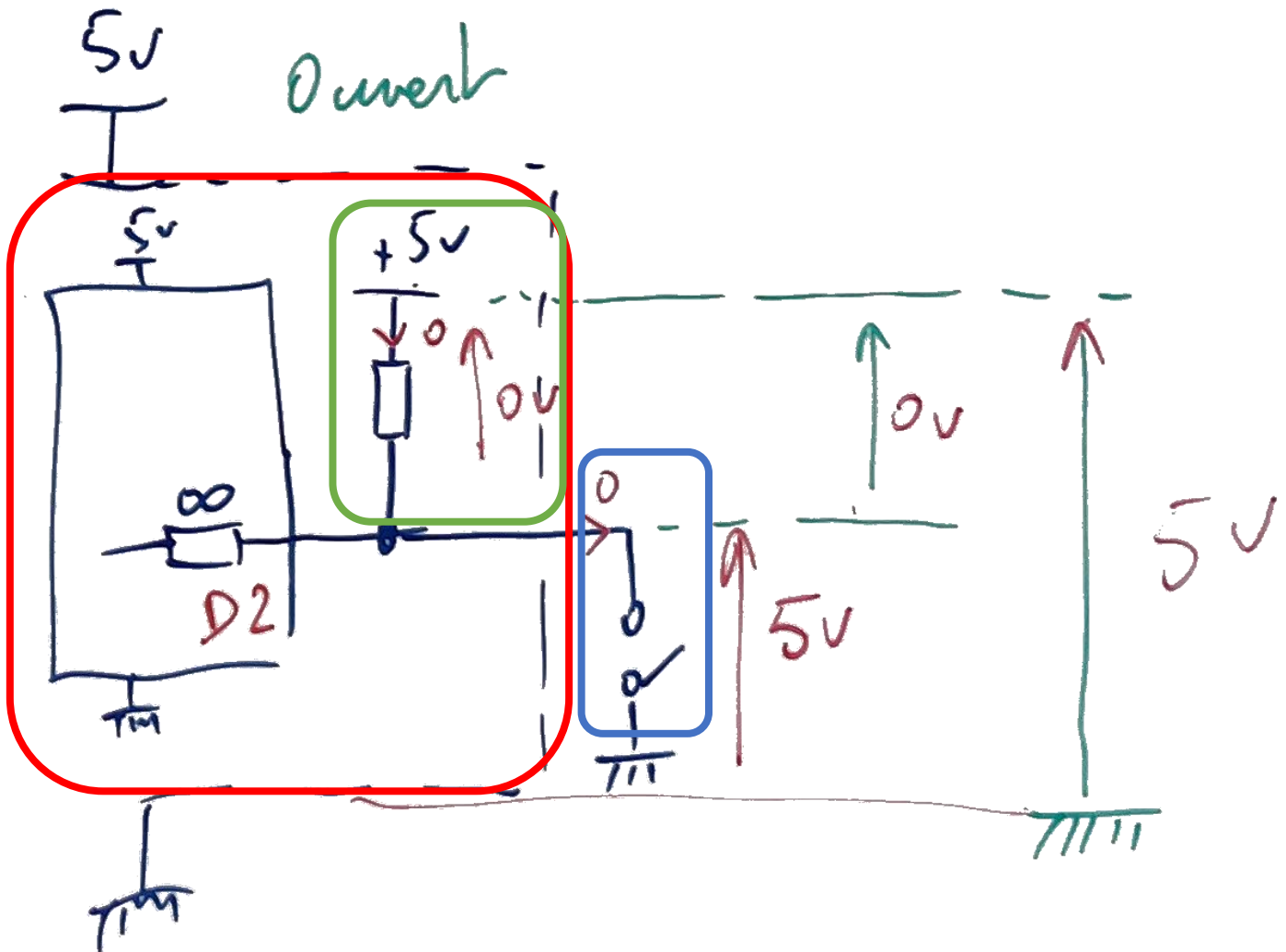


Figure 6 : Schéma de câblage du pluviomètre et de la carte Arduino (circuit ouvert)

Voici un tableau représentant les tensions mesurées par la carte Arduino en fonction de l'état du switch magnétique :

Etat du switch	Tension mesurée par la carte Arduino
Ouvert	5 V
Fermé	0 V

Figure 7 : Tableau des tensions mesurées par la carte Arduino

J'ai donc par la suite écrit un programme pour la carte Arduino permettant de capter les impulsions :

```
const byte interruptPin = 2;
const int interval = 500;
int nbImpulsion;
volatile unsigned long tiptime = millis();

void setup() {
  nbImpulsion = 0;

  Serial.begin(9600);

  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), count, FALLING);
}

void loop() {
}

void count() {
  unsigned long curtime = millis();

  if ((curtime - tiptime) > interval) {
    nbImpulsion = nbImpulsion + 1;
    tiptime = millis();
  }
}

void mesurer() {
  Serial.print(nbImpulsion);
  nbImpulsion = 0;
}
```

Figure 8 : Programme *pluviometre.ino*

Au debut de l'exécution du programme, le méthode `setup()` est lancée. Elle va d'abord initialisé la variable `nbImpulsion` à 0, il s'agit de la variable contenant le nombre de basculement entre chaque relevé de mesures qui auront lieu tout les certains temps en fonction de la période défini dans la base de données. Le débit de la communication série est défini à 9600 bauds. La résistance de pull-up est ensuite activé sur la pin 2. La méthode `count()` est ensuite définit pour être appelé à chaque chute de tension (basculement) de la pin 2.

A chaque basculement, une chute de tension est détecté par la carte Arduino, mais des bruits (composante non désirée) vont affecter le signal et que plusieurs impulsions soit lues au lieu d'une seule.

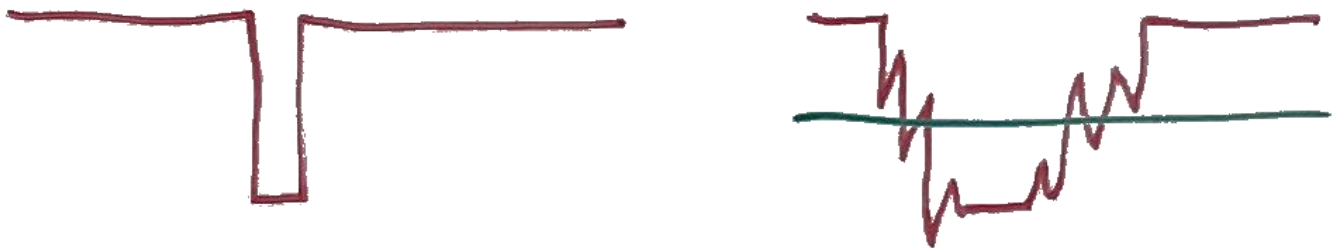


Figure 9 : A gauche, une impulsion « parfaite ». A droite, un impulsion avec du bruit.

Pour ne pas que le comptage d'impulsion soit faussé à cause des bruits, j'ai décidé de définir un interval de temps minimum à attendre entre chaque impulsions. J'ai défini cet interval à 500 millisecondes (`interval`). A chaque impulsions, la méthode `count()` va donc vérifier qu'aucune impulsions n'as été détectée dans les 500 millisecondes précédantes, puis va incrémenter la variable `impulsion`.

3.2) Communication avec la carte Arduino

La carte Arduino (zone en **rouge** sur la *Figure 10*) est reliée à une carte Raspberry (zone en **bleu** sur la *Figure 10*) par un cable USB. La méthode `mesurer()` est la méthode qui est appelé lors d'un relevé de mesures. A chaque appel, le nombre d'impulsions reçu (`impulsion`) est envoyé à la carte Raspberry via la communication série.

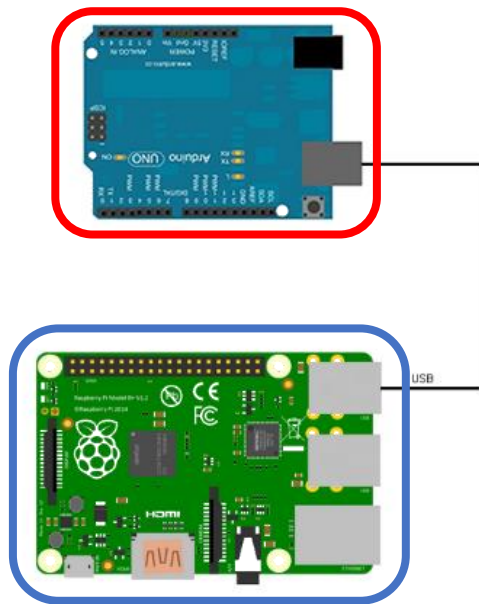


Figure 10 : Cablage carte Arduino et de la carte Raspberry

IV. Récupération des données de la base de données

4.1) Architecture de la base données

Voici le schéma entité relation de la base de données qui à été mis en place :

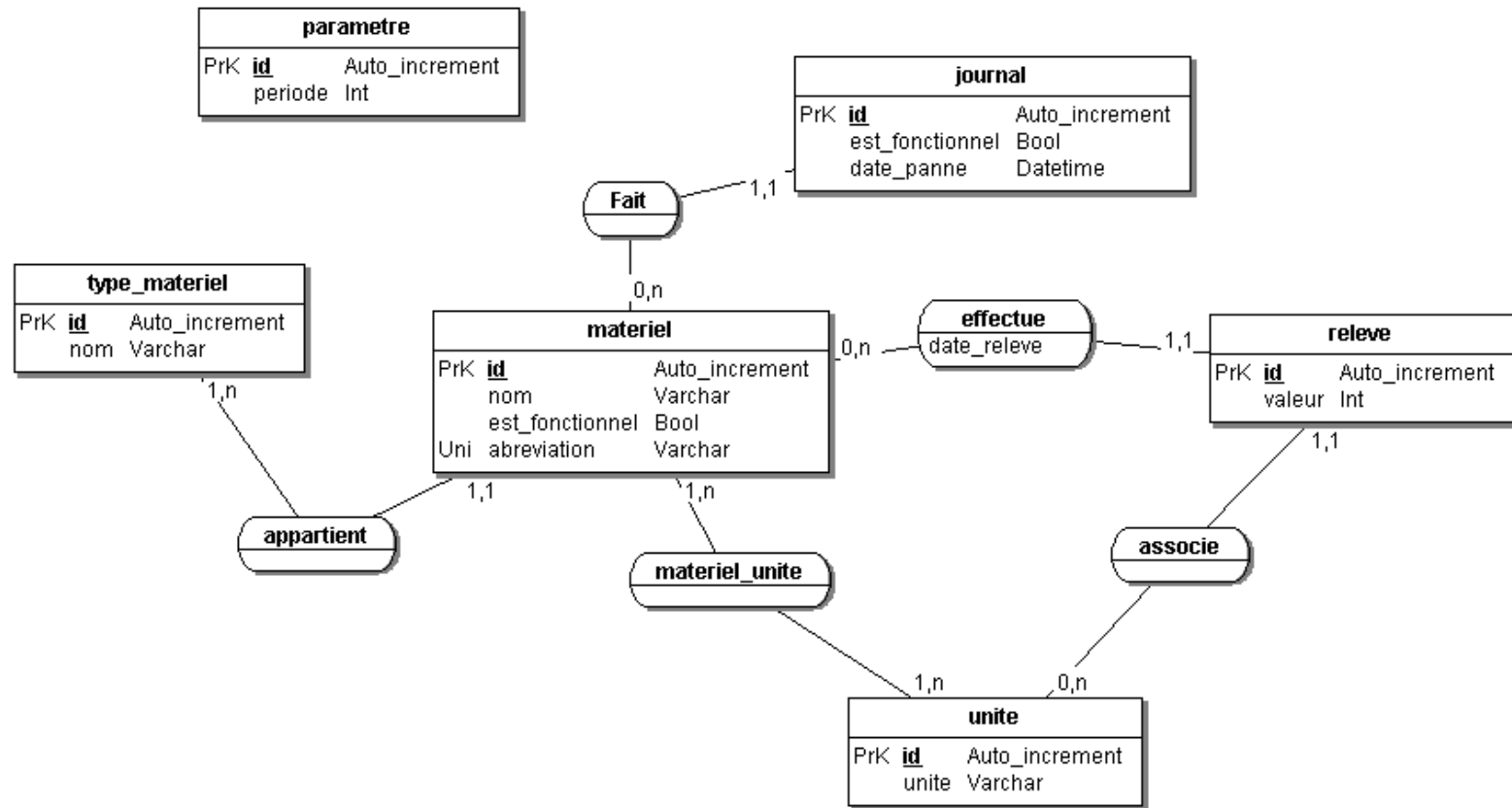


Figure 11 : Schéma entité relation de la base de données

4.2) Réalisation des pages PHP

Afin de pouvoir accéder aux enregistrements de la base de données depuis l'application Android, il a été nécessaire que je développe des page PHP. La première page PHP que j'ai développée à pour but de tester la connexion entre l'application Android et le serveur distant, je l'ai donc appelé `testConnexion.php`.

```
<?php
    echo ("OK") ;
?>
```

Figure 12 : Code de la page `testConnexion.php`

J'ai ensuite développé trois pages supplémentaires :

- `getSensors.php` : Cette page renvoi la liste des matériels de types « capteur » présent dans la base de données au format JSON.
- `getMicrocontrollers.php` : Cette page renvoi la liste des matériels de types « microcontrôleur » présent dans la base de données au format JSON.
- `getLogs.php` : Cette page renvoi la liste des journaux présent dans la base de données au format JSON.

Voici le code de la page `getSensors.php` :

```
<?php
    try
    {
        $bdd = new
PDO('mysql:host=localhost;dbname=supervision_serre;charset=utf8',
'projetbts', 'Nantes44');
        $reponse = $bdd->query('SELECT * FROM materiel WHERE id_type_materiel
= 1');
        $output = $reponse->fetchAll(PDO::FETCH_ASSOC);

        echo(json_encode($output));

        $reponse->closeCursor();
    }
    catch (Exception $e)
    {
        die('Erreur : ' . $e->getMessage());
    }
?>
```

Figure 13 : Code de la page `getSensors.php`

J'ai d'abord créé un nouvel objet de type PDO avec toutes les infos nécessaire (adresse, nom de la BDD, encodage, utilisateur), il s'agit de la connexion à la base de données. J'ai ensuite rédigé une requête qui retourne la liste des matériels de type « capteur ». La réponse de la requête est ensuite encodée en JSON avant d'être affichée.

Voici ce qu'affiche la page PHP :

```
[{"id":"1","nom":"Pluviometre DAVIS
7852M","abreviation":"pluvio","est_fonctionnel":"1","id_type_materiel":"1"},{
"id":"2","nom":"Anemometre
DAVIS","abreviation":"anemo","est_fonctionnel":"0","id_type_materiel":"1"},{
"id":"3","nom":"Solarimetre","abreviation":"solari","est_fonctionnel":"1","id_
type_materiel":"1"},{"id":"4","nom":"Sonde PT100 (eau
tuyau)","abreviation":"sonde_tuyau","est_fonctionnel":"1","id_type_materiel":
"1"},{"id":"5","nom":"Sonde PT100
(serre)","abreviation":"sonde_serre","est_fonctionnel":"1","id_type_materiel"
:"1"},{"id":"19","nom":"Potentiometre","abreviation":"potentio","est_fonction
nel":"0","id_type_materiel":"1"}]
```

Figure 14 : Affichage de la page `getSensors.php`

Pour les autres page PHP, j'ai seulement modifié la requête :

- `getMicrocontrollers.php` :

```
SELECT * FROM materiel WHERE id_type_materiel = 2
```

- `getLogs.php` :

```
SELECT journal.id, journal.est_fonctionnel, journal.date_panne, materiel.nom
FROM journal, materiel WHERE journal.id_materiel = materiel.id
```

V. Mise en place de l'application Android

5.1) Architecture de l'application

5.1.1) Les différentes activités

Sous Android, les différentes fenêtres d'IHM sont appelées des « activités » ou des « vues », elles permettent de rendre l'application plus fluide et plus intuitive. La première activité que j'ai développée est donc l'activité `MainActivity`. L'application Android Studio dispose d'un outil permettant de disposer les éléments à la main très simplement et rapidement, mais il est nécessaire perfectionner le code à la main (fichier XML) afin de l'optimiser et de le rendre compatible avec le plus d'appareil possible.

Voici à quoi ressemble l'activité `MainActivity` actuellement :

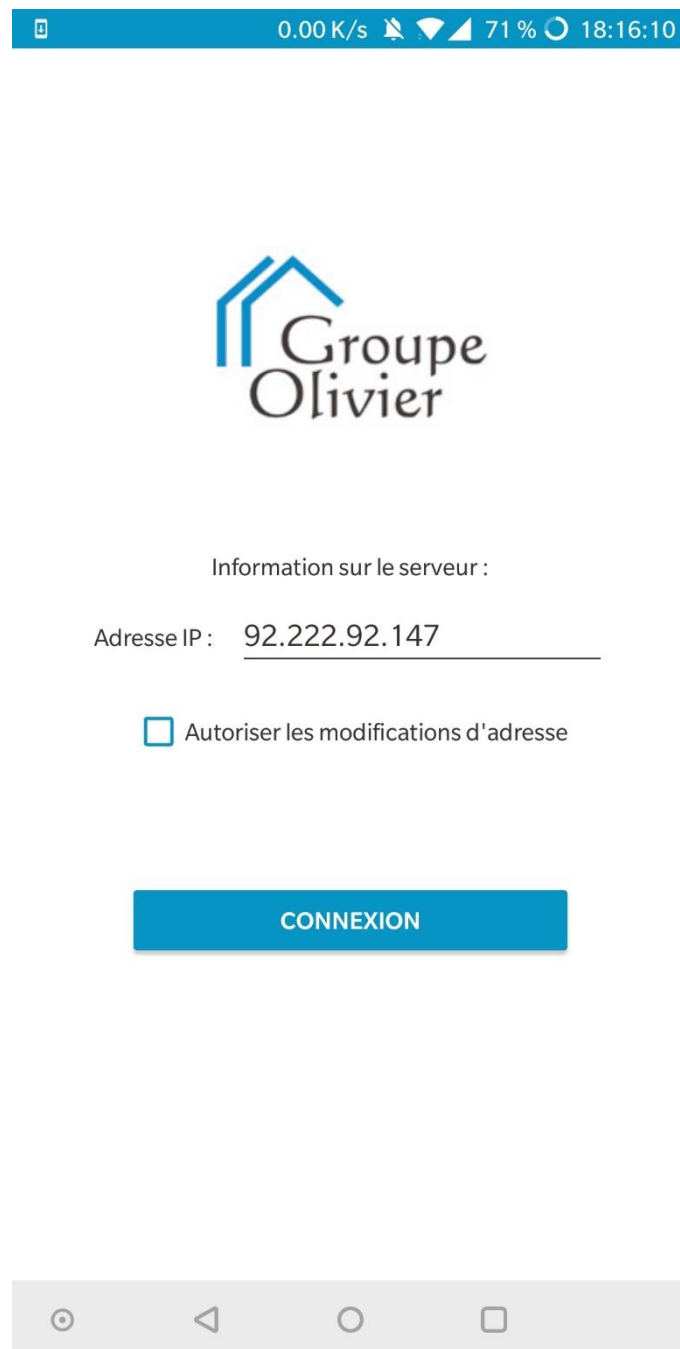


Figure 15 : Affichage de la page `MainActivity`

Après avoir cliqué sur le bouton « CONNEXION », l'utilisateur arrive sur l'activité `Dashboard` qui permet d'avoir une vue globale sur les différentes listes de matériel. Voici à quoi elle ressemble :



Figure 16 : Affichage de la page `DashboardActivity`

Comme vous pouvez le voir, j'ai également intégré un système d'affichage d'erreur qui permet d'être rapidement averti lorsqu'un problème est détecté. Le nombre important d'erreurs est dû au fait qu'au moment où j'ai lancé l'application, le montage n'était pas branché.

Lorsque l'utilisateur touche une des erreurs, une fenêtre pouvant afficher de plus amples détails s'ouvre.

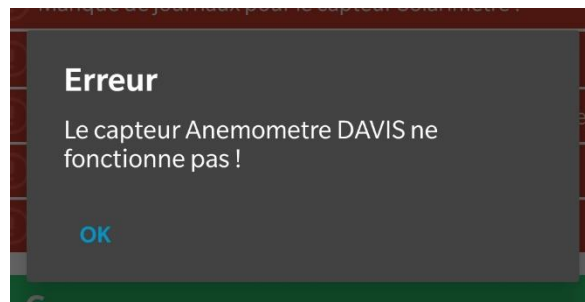


Figure 17 : Fenêtre d'erreur

Lorsque l'utilisateur touche une des tuiles (Capteurs, Microcontrôleurs ou Journaux) l'activité correspondante s'ouvre : `SensorsActivity`, `Microcontrollers` ou `LogsActivity`. Les activités `SensorsActivity` et `Microcontrollers` sont graphiquement identiques, voici à quoi elle ressemble (exemple avec `SensorsActivity`) :



Figure 18 : Affichage de la page `SensorsActivity`

A partir de cette activité, l'utilisateur a accès à la liste des capteurs ainsi que leur état de fonctionnement. Il est important de noter que chaque tuile correspondant à un capteur est générée dynamiquement, ce qui permet l'ajout de matériels sans avoir à modifier le code de l'application Android. Il peut également accéder au journal d'un capteur en le touchant. En effet, accéder à l'activité `LogsActivity` à partir de `SensorsActivity` ou `Microcontrollers` permet de filtrer et de n'afficher que le journal du matériel sur lequel l'utilisateur a cliqué :

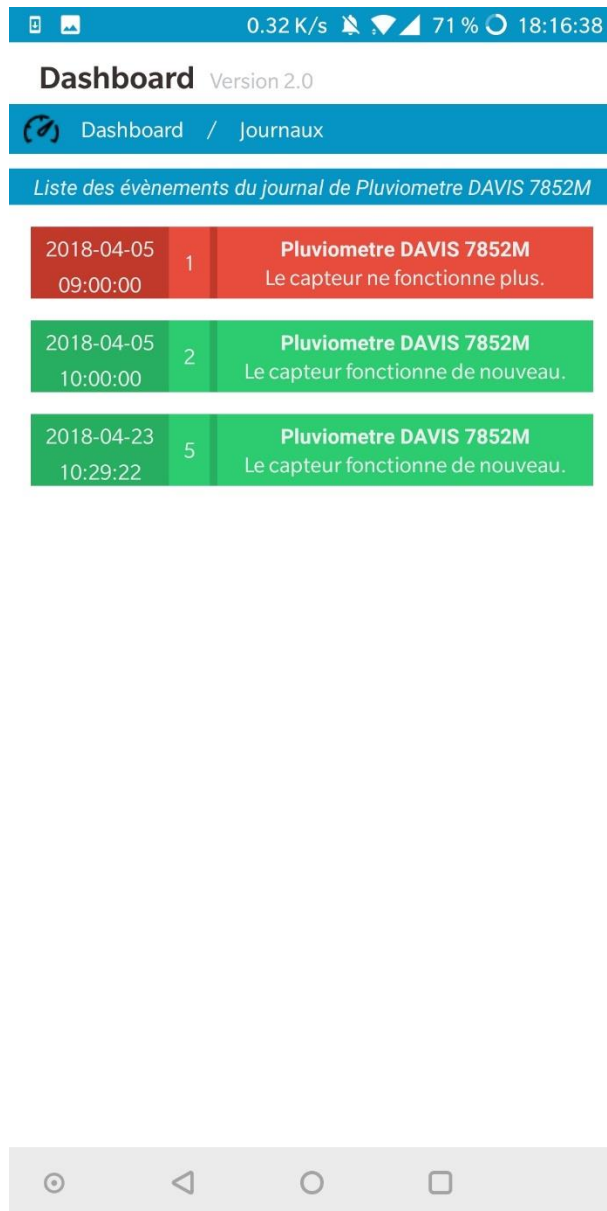


Figure 19 : Affichage de la page `LogsActivity` avec le filtre Pluviomètre DAVIS7852M

En accédant à l'activité `LogsActivity` à partir de `Dashboard`, la liste de tous les journaux serait affichée.

5.1.2) Les différentes classes

Afin de rendre le code le plus compréhensible et le plus réutilisable possible, j'ai développé l'application en ayant une approche orienté objet. Voici le diagramme de classe de l'application Android :

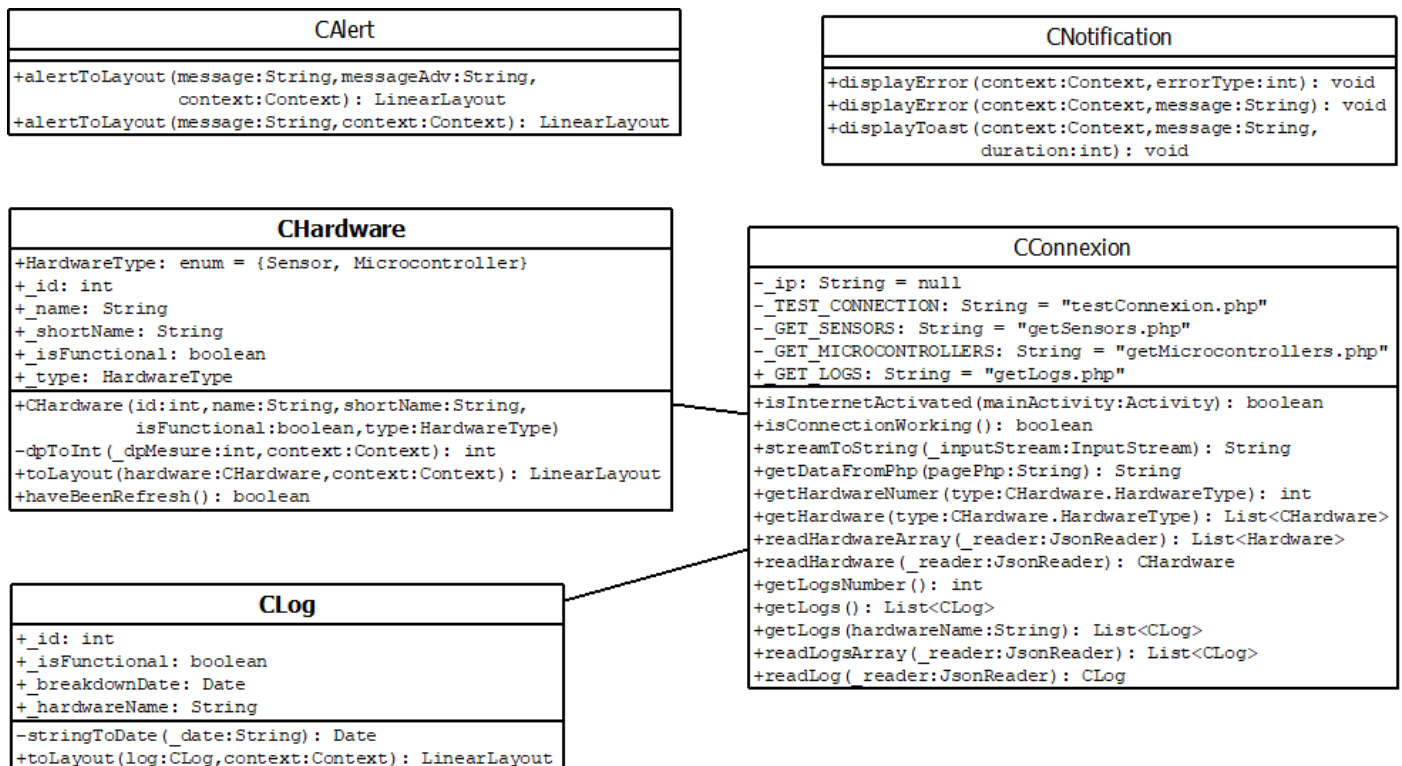


Figure 20 : Diagramme de classe de l'application Android

Classe	Utilité
<i>CHardware</i>	Chaque capteur et microcontrôleur sera représenté par objet de type CHardware.
<i>CLog</i>	Chaque évènement de journal sera représenté par un objet de type CLog.
<i>CConnexion</i>	Cette classe abstraite est utilisé pour communiquer avec les pages PHP.
<i>CNotification</i>	Cette classe abstraite est utilisé pour afficher des notifications à l'écran, comme « La connexion à réussis ».
<i>CAlert</i>	Cette classe abstraite est utilisé pour afficher les erreurs liées aux matériels comme vu sur la Figure 16.

5.2) Accès au pages PHP

Afin de récupérer les données issues des pages PHP, la classe CConnexion est utilisé. Elle possède les attributs suivants :

```

// ATTRIBUTES
private static String _ip = null;
private static final String _LOG_TAG = "CustomLog";
private static final String _TEST_CONNECTION = "testConnexion.php";
private static final String _GET_SENSORS = "getSensors.php";
private static final String _GET_MICROCONTROLLERS = "getMicrocontrollers.php";
private static final String _GET_LOGS = "getLogs.php";
  
```

Figure 21 : Attributs de la classe CConnexion

Comme vous pouvez le voir, elle possède un attribut correspondant à l'adresse IP du serveur distant. Cet attribut est initialisé lorsque le test de connexion au serveur (MainActivity) réussit. La classe CConnexion possède également en attribut le nom des différents pages PHP à consulter pour récupérer les données nécessaires.

Afin d'éviter certaines erreurs, avant d'effectuer une connexion avec une page PHP, il est préférable de tester si les itinérances de données sont activées, c'est le travail de la fonction `isInternetActivated()` :

```
/**
 * Renvoie vrai si l'itinérance de données du téléphone est activée
 * @param MainActivity
 * @return
 */
public static boolean isInternetActivated(Activity mainActivity) {
    ConnectivityManager cm = (ConnectivityManager) mainActivity.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    return netInfo != null && netInfo.isConnectedOrConnecting();
}
```

Figure 22 : Fonction `isInternetActivated()` de la classe `CConnexion`

Une fois que la vérification à été faite, il est alors possible d'accéder aux pages PHP du serveur. Voici la fonction `isConnectionWorking()` qui permet de tester la connexion entre le smartphone et le serveur distant :

```
public static boolean isConnectionWorking() throws InterruptedException, ExecutionException {
    ExecutorService executor = Executors.newSingleThreadExecutor();
    Callable<Boolean> callable = () -> {
        InputStream inputStream = null;
        try {
            final HttpURLConnection conn = (HttpURLConnection) new URL("http://" + _ip + "/" + _TEST_CONNECTION).openConnection();
            conn.setReadTimeout(10000);
            conn.setConnectTimeout(15000);
            conn.setRequestMethod("GET");
            conn.setDoInput(true);
            conn.connect();
            inputStream = conn.getInputStream();
            String data = streamToString(inputStream);
            data = data.substring(0, data.length()-1);
            if (data.equals("OK")) {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) {
            Log.e(_LOG_TAG, e.getMessage());
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
        }
        return false;
    };
    Future<Boolean> future = executor.submit(callable);
    executor.shutdown();
    return future.get();
}
```

Figure 23 : Fonction `isConnectionWorking()` de la classe `CConnexion`

Cette fonction crée un nouveau Thread afin de ne pas bloquer l'application. Elle crée ensuite un objet de type `HttpURLConnection` auquel elle associe le lien de la page PHP, ainsi que plusieurs paramètres permettant entre autres de définir un délai maximum à attendre avant de déclarer que la connexion à échoué. Elle récupère ensuite les données de la page, puis vérifie si les données sont égales à « OK », si oui la connexion à fonctionner.

Le principe est le même pour les autres pages, mais au lieu de retourner un boolean, les fonctions retournent les données de la page qui sont donc au format JSON dans une chaîne de caractère.

5.3) Décoder le JSON

Etant donné que les fonctions retournent des données au format JSON, il est nécessaire des les décoder afin de pouvoir instancier des objets pour chaque matériel ou évènement de journal.

Par exemple, pour récupérer la liste des capteurs, il faut appeler le fonction `getHardware()` de la classe `CConnexion` en passant en paramètre le type de matériel recherché, dans notre cas `CHardware.HardwareType.Sensors`.

```
/**
 * Affiche les capteurs
 */
private void displaySensors() {
    try {
        sensors = CConnexion.getHardware(CHardware.HardwareType.Sensor);
        for (final CHardware sensor : sensors) {
            LinearLayout sensorLayout = CHardware.toLayout(sensor, context: this);
            sensorLayout.setOnClickListener((v) -> {
                Intent logActivityIntent = new Intent( packageContext: SensorActivity.this, LogActivity.class);
                logActivityIntent.putExtra( name: "EXTRA_FILTER", sensor.get_name());
                startActivity(logActivityIntent);
            });
            layout_sensors.addView(sensorLayout);
        }
    } catch (Exception e) {
        Log.i( tag: "CustomLog", e.getMessage());
    }
}
```

Figure 24 : Fonction `displaySensors()` de l'activité `SensorActivity`

Voici le code de la fonction `getHardware()` :

```
public static List<CHardware> getHardware(CHardware.HardwareType type) throws IOException, Int
    JsonReader reader;
    switch (type) {
        case Sensor:
            reader = new JsonReader(new StringReader(getDataFromPhp(_GET_SENSORS)));
            break;
        case Microcontroller:
            reader = new JsonReader(new StringReader(getDataFromPhp(_GET_MICROCONTROLLERS)));
            break;
        default:
            return null;
    }
    try {
        return readHardwareArray(reader);
    } finally {
        reader.close();
    }
}
```

Figure 25 : Fonction `getHardware()` de la classe `CConnexion`

Comme vous pouvez le voir, en fonction du type passé en paramètre, les données sont récupérées sur une page PHP différentes. Les données sont récupérées à l'aide d'un `JsonReader`, puis passées en paramètre lors de l'appel de la fonction `readHardwareArray()` qui va permettre de différencier les différents objets.

```
private static List<CHardware> readHardwareArray(JsonReader _reader) throws IOException {
    List<CHardware> hardwareList = new ArrayList<>();
    _reader.beginArray();
    while (_reader.hasNext()) {
        hardwareList.add(readHardware(_reader));
    }
    _reader.endArray();
    return hardwareList;
}
```

Figure 26 : Fonction `readHardwareArray()` de la classe `CConnexion`

Tout d’abord, une liste de matériel est déclarée et initialisée. Ensuite, pour chaque objet présent dans les données JSON passées en paramètre, la fonction `readHardware()` est appelée, afin de les instancier en objets `CHardware` avec les attributs correspondants.

```
private static CHardware readHardware(JsonReader _reader) throws IOException {
    int id = -1;
    String name = null;
    String shortName = null;
    boolean isFunctional = false;
    CHardware.HardwareType type = null;

    _reader.beginObject();
    while (_reader.hasNext()) {
        String tag = _reader洗nextName();
        if (tag.equals("id")) {
            id = _reader.nextInt();
        } else if (tag.equals("nom")) {
            name = _reader.nextString();
        } else if (tag.equals("abreviation")) {
            shortName = _reader.nextString();
        } else if (tag.equals("est_fonctionnel")) {
            isFunctional = (_reader.nextInt() > 0);
        } else if (tag.equals("id_type_materiel")) {
            type = CHardware.HardwareType.values()[_reader.nextInt()-1];
        } else {
            _reader.skipValue();
        }
    }
    _reader.endObject();
    return new CHardware(id, name, shortName, isFunctional, type);
}
```

Figure 27 : Fonction `readHardware()` de la classe `CConnexion`

Ainsi, chaque balises présentes dans les données JSON passées en paramètres sont associées à un attribut de la classe `CHardware`. La fonction retourne donc un nouvel objet correspondant aux données passées en paramètres. La fonction `getHardware()` appelée en premier lieu va donc retourner la liste des objets demandées.

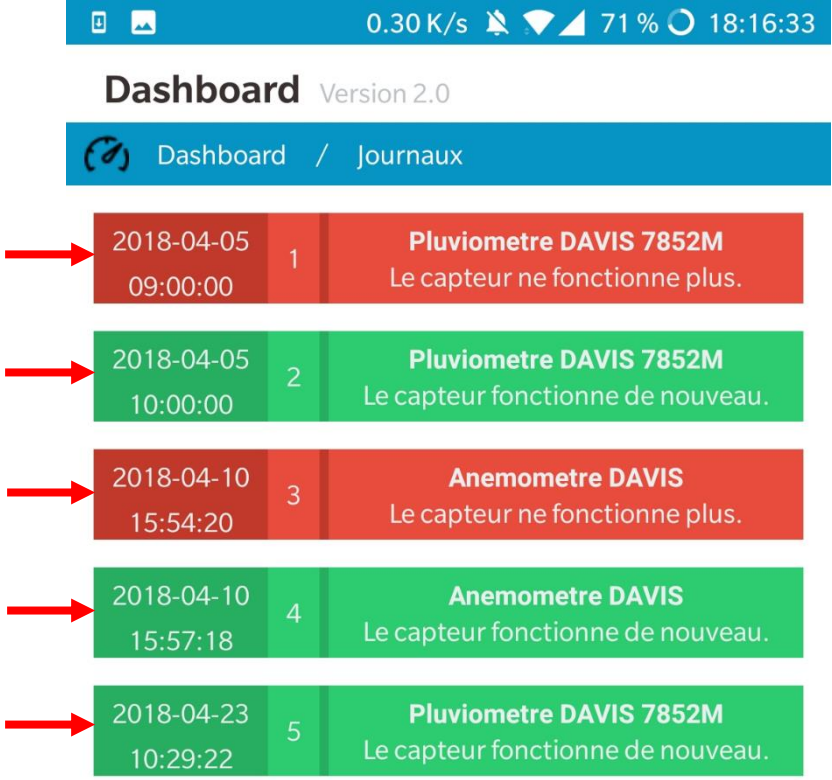
VI. Tests fonctionnels

Pour tester certaines méthodes, j'ai effectué des tests fonctionnels.

6.1) Méthode getHardwareNumber()

Test CConnexion.getHardwareNumber()	
Résultats attendus	
Récupérer le nombre de capteurs présent dans la base de données : 6	
Validation	
<ul style="list-style-type: none">- Ouvrir l'application sur un smartphone Android- Toucher le bouton « Connexion »- Vérifier que l'application affiche le nombre 6 dans la tuile « Capteurs »	
Résultat dans la console	
	

6.2) Méthode getLogs()

Test CConnexion.getLogs()	
Résultats attendus	
Récupérer les évènements de journaux présents dans la base de données. Il y en a cinq.	
Validation	
<ul style="list-style-type: none"> - Ouvrir l'application sur un smartphone Android - Toucher le bouton « Connexion » - Toucher la tuile « Journaux » - Vérifier que l'application affiche les cinq évènements de journaux. 	
Résultat dans la console	
	

VII. Conclusion

Pour conclure, je dirais que les tâches qui m'ont été attribuées ont été très intéressantes. Grace à la tâche qui consistait à communiquer avec le pluviomètre, j'ai pu mettre en pratique les connaissances sur les systèmes embarqués que j'ai acquis tout au long de mon parcours scolaire ainsi que par des projets personnels. Grace à cette tâche j'ai également beaucoup appris au niveau des circuits électriques. Il a fallu être prudent sur les différentes tensions afin de ne pas endommager les capteurs ainsi que les cartes Arduino et Raspberry.

Ma deuxième tâche qui consistait à développer l'application Android a également été très enrichissante. Par le passé j'avais déjà utilisé l'IDE Android Studio, mais sa stabilité laissait à désirer. Cette tâche m'a permis de prendre connaissance de l'évolution de cet outil, et de reprendre du plaisir à développer des applications Android.

Etant donné que je compte me diriger vers des études de développement logiciel, la conception et le développement de cette application Android m'ont permis d'acquérir des connaissances qui me seront très utiles.