POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

RAPPORT DE PROJET : LA GAMELLE CONNECTÉE

DAMIENS Lutécia

DORIAN Alexis

Département IMA

2016-2017

Tuteurs :

BOE Alexandre

VANTROYS Thomas

REDON Xavier

Remerciements

Tout d'abord, nous tenons à remercier l'équipe pédagogique de notre formation Informatique, Microélectronique et Automatique (IMA) qui nous a enseigné les savoirs et connaissances utiles et nécessaires pour la réalisation de ce projet.

Ensuite, nous remercions également nos tuteurs de projet, Monsieur Alexandre BOE, Monsieur Xavier REDON et Monsieur Thomas VANTROYS pour leur aide et conseils tout au long du projet.

De plus, nous souhaitions remercier les responsables du fabricarium pour leur aide et leurs conseils sur la conception mécanique de notre projet.

Enfin, un merci particulier à Monsieur Laurent ENGELS pour le temps qu'il nous accordé pour la réalisation de notre vidéo.

Sommaire

Introduction

Dans le cadre de notre projet de 4ème année de la formation ingénieur à Polytech Lille, en Informatique, Microélectronique et Automatisme, nous avons réalisé un projet tout au long du semestre 8. Notre projet est le suivant : la gamelle connectée.

L'internet des objets (IoT) représente l'extension d'internet à des choses et à des lieux du monde physique. Les objets connectés en l'occurrence sont aujourd'hui un enjeu important. De plus en plus d'objets initialement banals se voient être connectés afin d'en améliorer l'utilisation. Lors de ce projet, nous nous intéressons aux animaux domestiques. Lorsque nous nous absentons, nous n'avons aucun contrôle sur ce qui se passe à la maison. Ainsi, nous ne pouvons pas savoir si notre animal mange ou se porte bien. De plus, nous n'avons aucun retour sur ce que notre chien ou chat notamment a consommé. Est-ce qu'il se nourrit bien ? À quelle fréquence ? Dès lors, il est impensable de s'absenter sur le long terme ou un week-end sans faire intervenir un tiers pour nourrir notre cher animal. Nous pouvons également penser à des animaux qui seraient dangereusement en surpoids. Nous voudrions surveiller leur alimentation mais cela demande une surveillance constante qui n'est pas forcément envisageable. L'objectif de ce projet est de répondre à ces questions et besoins. Et si nous connections une gamelle à une application ?

Dans ce rapport, nous commencerons par faire une présentation détaillée du projet. Puis, nous vous décrirons la mécanique de l'objet ainsi que la partie arduino. Ensuite, nous vous parlerons d'avantage de l'aspect connecté en explicitant la connexion entre le module wifi et l'application mobile. Nous reviendrons alors en détails sur cette dernière.

I. Présentation du projet

a. Objectif et cahier des charges

L'objectif de ce projet est de fabriquer une gamelle pour animaux connectée. Pour ce faire, trois grands axes sont à dégager : une partie informatique, électronique et mécanique. Un système mécanique doit être réfléchi et réalisé afin de créer un réservoir qui alimente la gamelle. Pour la mesure du poids contenu dans la gamelle, un choix technologique devra être réalisé. Soit une gamelle alimentaire sera recyclée, soit l'établissement d'un dosage en volume sera effectué. L'objet connecté sera contrôlé par un arduino et l'utilisation d'un module wifi sera réalisée afin de pouvoir contrôler le système depuis l'application mobile. Des capteurs devront être utilisés telles que des capteurs infrarouges pour vérifier si le réservoir est vide ou non, ou encore des capteurs de présence afin de vérifier l'absence de l'animal pour la prise de décisions. Côté électronique, il faudrait idéalement rendre l'objet autonome et de l'alimenter par batterie ou secteur. Le réapprovisionnement automatique de la gamelle est très important.

b. Choix techniques : matériel et logiciel

Tout d'abord, nous avons réalisé l'état de l'art des gamelles connectées déjà réalisées et commercialisées. Vous trouverez cet état de l'art en Annexe 1.

En ce qui concerne le dosage de la gamelle, nous notons que beaucoup d'objets utilisent les volumes afin de réaliser la mesure du poids. De plus, l'alimentation est souvent sur batterie et parfois sur secteur. D'un point de vue connecté, les systèmes sont tous reliés au WiFi de la maison mais nous n'en savons pas d'avantage sur les technologies utilisées puisqu'il s'agit bien souvent de systèmes brevetés.

Ensuite, en ce qui concerne nos choix techniques, nous avons choisi de réaliser un système constitué d'une hélice contrôlée par un servomoteur branché sur un arduino méga. Pour ce faire, le réservoir sera en forme « pyramidale inversée » pour permettre aux croquettes de glisser naturellement. Le servomoteur fera tourner le système d'hélice et déversera les croquettes dans la gamelle. Un système de dosage en volume est alors utilisé. Le contrôle du réservoir et de la gamelle se feront alors de la même manière. Un dispositif LED Infrarouge – phototransistor, fonctionnant en tout ou rien, permettra de savoir si la gamelle est vide ou non. Dans le réservoir, ce dispositif sera mis à différents niveaux afin d'informer l'utilisateur si la gamelle est à moitié vide par exemple (ou à moitié pleine).

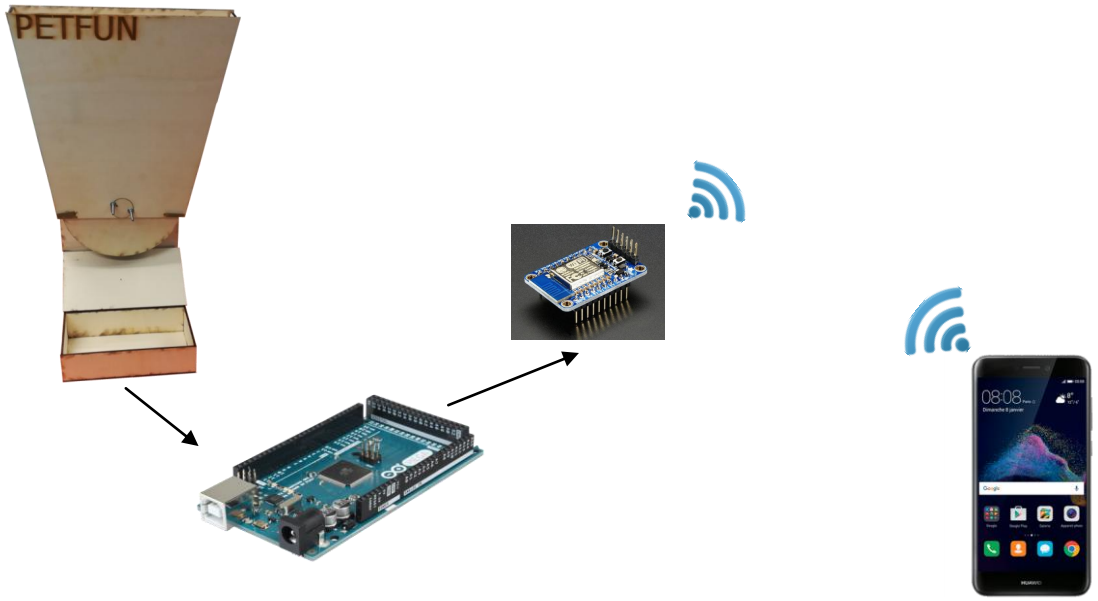
De plus, des capteurs ultrasons seront utilisés afin de détecter la présence de l'animal devant la gamelle. L'alimentation se fera sur secteur grâce à un adaptateur 5V.

Enfin, du côté de l'arduino, un module WiFi ESP8266 sera utilisé afin de connecter l'objet au WiFi de la maison. Une carte shield SD servira à stocker les valeurs des capteurs et de l'état de la gamelle. Un module temps réel (RTC) sera également nécessaire afin de pouvoir programmer en fonction de l'heure qu'il est. (Si l'utilisateur souhaite déverser des croquettes à 11:30, le programme arduino doit être capable de répondre à cette requête en temps réel.

Voici donc la liste de matériel réalisée en conséquence :

Composant	Lien	Checklist	Quantité
Emetteur + récepteur IR SEN00241	http://www.gotronic.fr/art-emetteur-recepteur-ir-sen00241-25499.htm	IR SFH4554 x2 Matériel de M. Redon	6-7
Adaptateur PS512S	http://www.gotronic.fr/art-adaptateur-ps512s-19913.htm	Commandé	1
Shield carte SD V4 103030005		Shield SD ethernet + carte SD Matériel de M. Redon	1
Module horloge temps réel ADA3296	http://www.gotronic.fr/art-module-horloge-temps-reel-ada3296-25536.htm	Commandé	1
Capteur Ultrason		Matériel de M. Redon	1
LEDs		Matériel de M. Redon	5
Résistances pour LEDs et pont diviseur de tension pour les IR/phototransistor		Salle C205	x
Arduino Méga		Matériel de M. Redon	1
ESP8266		Matériel de M. Redon	1
Servomoteur 360°		Matériel de M. Redon	1
Solution de secours / amélioration : raspberry et balance à trafiquer			1

Le schéma global de connexion du système est le suivant :



L'arduino mega qui est relié à tous les capteurs et *shields*, est connectée à l'ESP8266 via du 3.3V. Le module WiFi est connecté sur le WiFi local (de la maison). L'application mobile, connectée au WiFi, se connecte au serveur TCP contenu sur le module WiFi.

II. Partie mécanique et arduino

a. La mécanique

Tout d'abord, la modélisation du réservoir et du système d'hélice ont été réalisés sous *Solidworks*. Pour ce faire, nous avons dû décider de combien de grammes de croquettes à la fois nous pouvons réapprovisionner la gamelle. Ainsi, nous avons effectué des recherches quant à la consommation de nourriture journalière d'un chat. Étant donné qu'un chien consomme davantage de croquettes qu'un chat, nous nous basons sur la consommation de ce dernier, quitte à donner plusieurs doses si l'animal est un chien. Voici un tableau sur lequel nous allons nous baser :

Poids chat adulte	1-3kg	4kg	5kg	6kg
Quantités journalières (kg)	15-50g	50-65g	65-70g	70-90g
Quantités journalières (300 mL)	X 0.1 – 0.3	X 0.3 – 0.4	X 0.4	X 0.4-0.6

Ce tableau marche pour les croquettes « Ultra premium direct » en particulier.

Grâce à ce tableau, nous pensons que réapprovisionner la gamelle par dose de 15g est approprié.

Or, $15g = 30mL = 3 \times 10^{-5}\ m^3$

Maintenant, il est nécessaire de calculer le rayon de l'hélice permettant au système de doser 15g à la fois.

Nous voulons pouvoir doser 15g : $\frac{\pi R^2 h}{n} = 3 \times 10^{-5}\ m^3$

Posons n = 4 car nous souhaitons avec 2 doseurs, un de chaque côté de l'hélice :

$$\pi R^2 h = 12 \times 10^{-5}\ m^3$$

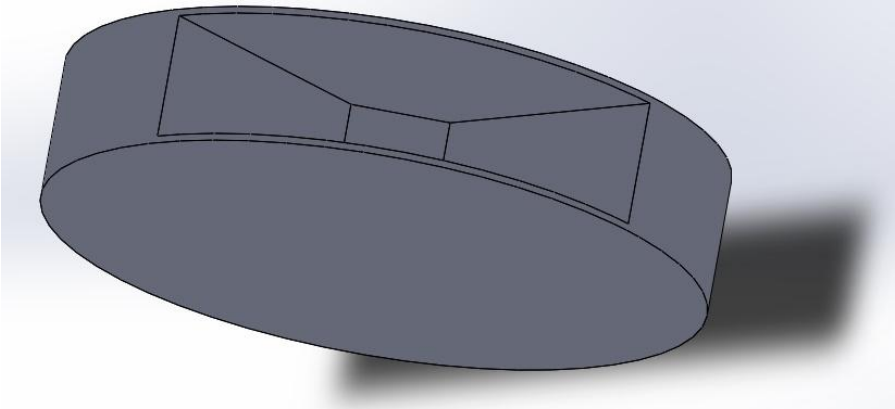
Prenons à présent h = 4 cm ; h étant la largeur de l'ouverture conduisant à la gamelle. Nous avons alors :

$$\pi R^2 = 3 \times 10^{-2}\ m^3$$

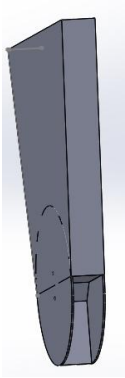
$$R^2 = \frac{3 \times 10^{-2}}{\pi}\ m^2$$

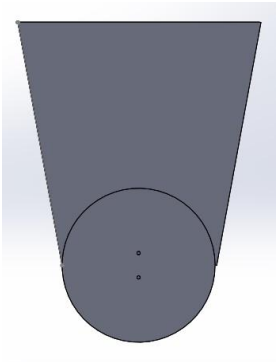
$$\boxed{R \approx 9{,}5\ cm}$$

Voici le système d'hélice modélisé :



Et voici l'assemblage réservoir et système hélice :





La gamelle a été entièrement réalisée à la découpeuse laser, en bois. Afin de réaliser le système d'hélice, une mise en plan de la modélisation sous Solidworks a été réalisée. Ainsi, nous avons collé les différentes épaisseurs afin de réaliser la gamelle.

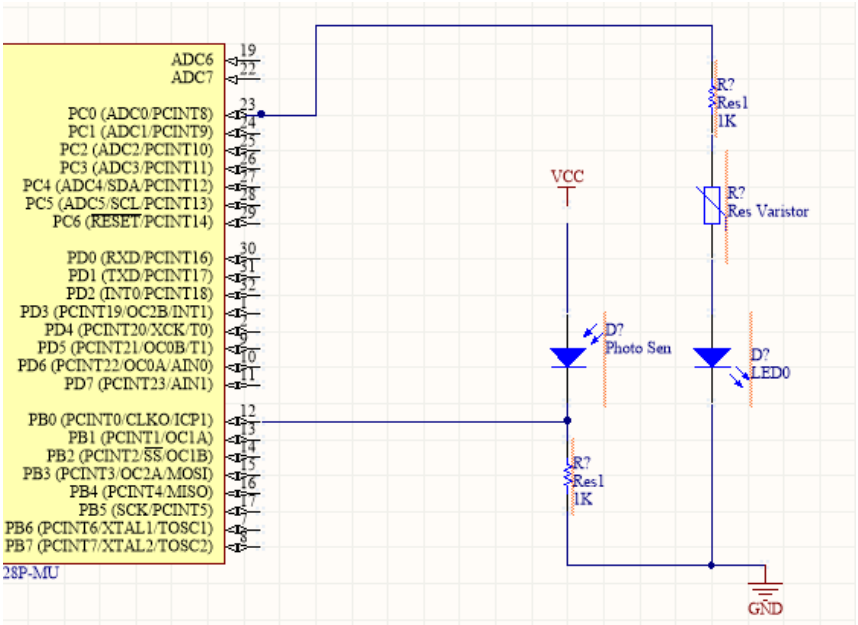
Le reste de notre objet connecté a été réalisé sous inkscape. Voici l'aspect de notre objet à la fin du projet :

b. Les composants connectés à l'arduino

i. Les capteurs et le moteur

1. LED infrarouge-phototransistor

Le système LED infrarouge-phototransistor sera installé dans la gamelle et dans le réservoir (à différents niveaux). Cela permettra de savoir si la gamelle est pleine et de connaitre le niveau du réservoir. Ce système fonctionne en tout ou rien. Soit l'onde traverse et atteint le récepteur, soit non. Pour ce faire, nous utilisons une LED IR SFH 4554 pour l'émission et une pour la réception. Pour des raisons de simplicité de fonctionnement, l'allumage des LEDs ne sera pas piloté. Elles seront continuellement allumées (aucune information n'est à transmettre). À la réception, nous utilisons une photodiode en pull-down avec une résistance pour contrôler l'intensité dans cette dernière. Nous relions ce montage à un arduino pour vérifier jusqu'à quelle portée nous détectons le signal de la LED émettrice. Grâce à cela, nous serons à même de définir les emplacements des émetteurs/récepteurs dans la gamelle et dans le réservoir ainsi que la taille de ces derniers. Voici le schéma de principe :



Avant de réaliser le montage, les résistances du schéma doivent être calculées. En réception, une résistance de 4.7 kΩ est placée. En émission, nous alimentons la LED en 5 V. Elle possède un courant U_d = 1.2 V pour 20 mA et supporte 100 mA continue. À 100 mA, son U_d = 1.4 V. Pour être certains d'avoir la meilleure portée possible, nous nous rapprochons des 100 mA. Ainsi, pour le dimensionnement de R nous obtenons:

$$R = \frac{V_{CC} - U_d}{I_f}$$

A.N. : $V_{CC} = 5\,V$; $U_d = 1.4\,V$; $I_f = 0.1\,A$
$R = 36\,\Omega$
En normalisant, $\boxed{R = 47\Omega}$
Avec ces données, nous obtenons une portée maximale de 30 cm.

Nous faisons un autre test avec une intensité plus basse :
A.N. : $V_{CC} = 5V$; $U_d = 1.2\,V$; $I_f = 0.01\,A$
$R = 380\,\Omega$
En normalisant, $\boxed{R = 330\Omega}$
Avec ces données, nous obtenons une portée maximale de 22 cm.

Ces valeurs nous ont permis de dimensionner correctement la gamelle avant de la réaliser.

2. Capteur Ultrason et capteur infrarouge (présence)

Un capteur ultrason est nécessaire afin de savoir si l'animal est devant la gamelle. En effet, si l'animal est en train de s'alimenter, les retours des capteurs peuvent être faussés par sa présence. Le module HC-SR04 est celui que nous avons à notre disposition. L'étude de ce capteur étant réalisée, nous nous sommes rendu compte que les animaux, et notamment le chat, étaient capables d'entendre les ultrasons. Ce problème nous a poussés à ne finalement pas utiliser ce capteur car l'animal pourrait être dérangé et ne jamais s'approcher de la gamelle. La solution que nous avons alors adoptée pour la suite pour tenir ce rôle est alors le capteur infrarouge IRS05A.

Afin de réaliser le programme en C correspondant, nous avons étudié la *datasheet* du composant et en particulier :

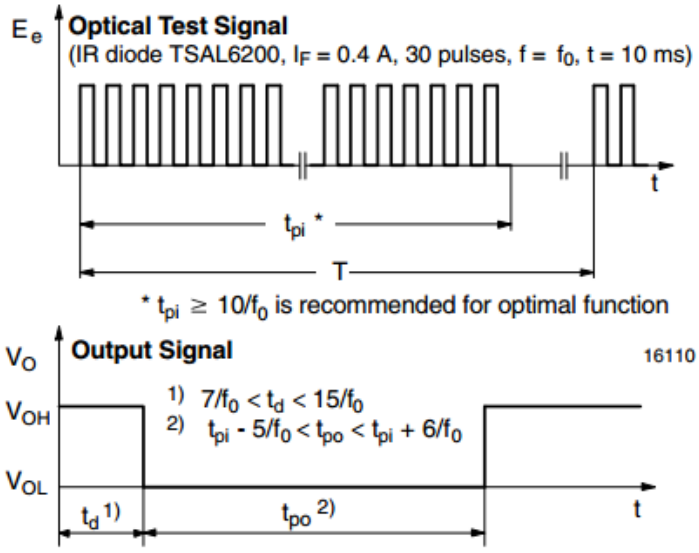
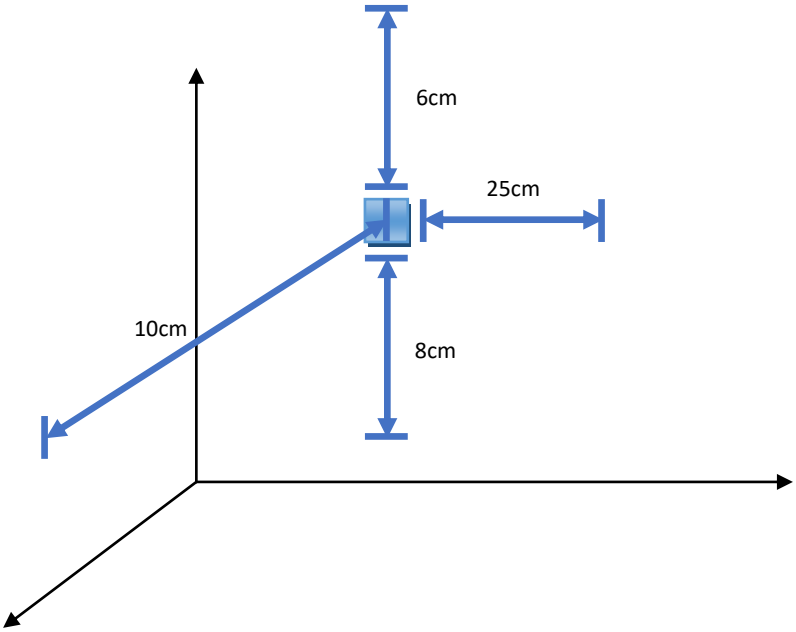
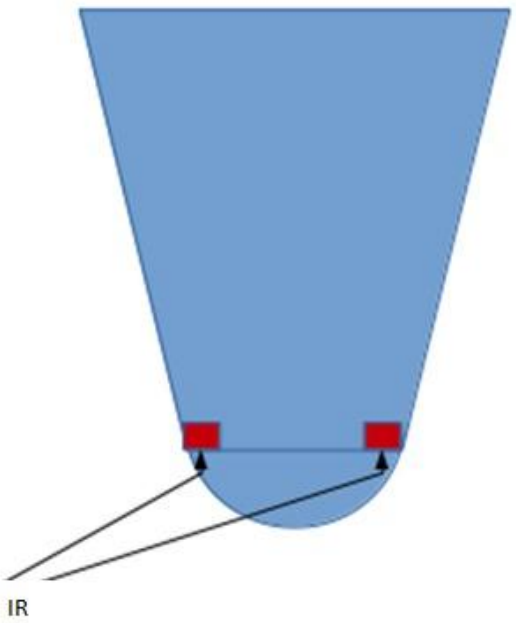


Fig. 1 - Output Active Low

Pour l'expérimentation, nous avons juste besoin de commander l'entrée *enable* et de placer une LED à la sortie *output* afin de visualiser si le module détecte ou non un objet. Suite à cette expérimentation, nous avons pu définir la zone de détection du module par la pratique. Voici un schéma 3D décrivant la portée du dispositif :



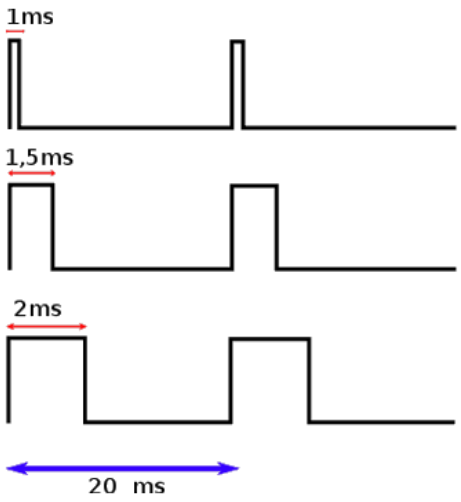
D'un point de vue pratique, nous avons donc choisi de tenir compte de la portée du capteur mesurée. Ainsi, voici un schéma représentant ces emplacements :



3. Servomoteur

Le servomoteur que nous utilisons est un servomoteur à courant continu. Il a l'avantage de pouvoir être contrôlé en vitesse et de pouvoir tourner à plus de 180°.

Pour alimenter un servomoteur, il faut brancher ses trois connectiques sur l'alimentation 5V de l'arduino, sur la masse et sur une broche PWM. En effet, la consigne envoyée à un servomoteur est de type PWM (*Pulse Width Modulation*). Ce signal permet de faire varier la durée en état « haut » du signal grâce à la consigne que nous lui donnons. Voici un schéma montrant quelques exemples de modifications de la durée d'un état « haut » :



La difficulté majeure a été de programmer ce servomoteur en passant par le langage C. En effet, avec arduino IDE, il est plus aisé de le faire puisqu'il existe une bibliothèque *servo.h* permettant de programmer des servomoteurs facilement grâce à l'objet *servo*. En C, il a fallu se pencher sur le fonctionnement et le contrôle d'une PWM grâce aux différents registres de l'arduino.

Dans un premier temps, il faut configurer les registres TCCR1A et TCCR1B afin d'activer la PWM en mode non-inversé et à fréquence élevée.
Enfin, il faut activer la PIN5 du port B, puisque nous l'avons choisi, en sortie (Pin PWM).

Rentrons un peu plus dans le détail quant à la configuration des deux registres TCCR1A et TCCR1B. Nous avons TCCR0A = 1000 0001 = 0x81 et TCCR0B = 0000 0101 = 0x05. Afin de comprendre ces configurations, il est nécessaire de consulter les tableaux de la *datasheet* correspondants.

Les voici :

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM (inverting mode)

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

Table 16-8. Waveform Generation Mode Bit Description

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

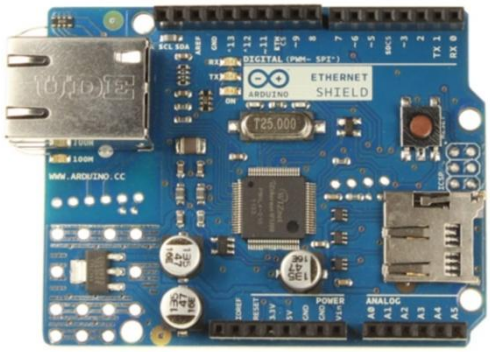
TCCR0A : COM0A1 = 1 COM0A0 = 0 => non-inverting mode
COM0B1 = 0 COM0B0 = 0 => mode standard
WGM01 = 0 WGM00 = 1 => PWM mode

TCCR0B : FOC0A = 0 FOC0B = 0 => car PWM mode
WGM02 = 0 => PWM mode
SC02 = 1 CS01 = 0 CS00 = 1 => clk (entrée/sortie) /1024

Dans un deuxième temps, par simple modification du registre OCR0A, nous agissons directement sur la commande du servomoteur. La durée en état « haut » est ainsi proportionnelle à la consigne saisie dans OCR0A. Le montage réalisé avec le système d'hélice contrôlée par le servomoteur présentant encore certains frottements, nous avons essayé différentes commandes dans OCR0A afin d'alimenter la gamelle convenablement, c'est-à-dire, permettre à la gamelle d'effectuer un demi-tour.

ii. Le shield SD

Pour rendre notre projet innovant par rapport aux autres gamelles connectées proposées sur le marché, nous avons décidé de faire un suivi lié à l'alimentation du chat. Par exemple, nous souhaitons savoir à quels moments de la journée l'animal va manger, pendant combien de temps, en quelles quantités etc… Pour cela, nous devons stocker ces différentes données. Pour ce faire, nous utiliserons le shield SD/ethernet suivant :



Pour commencer, l'expérimentation du shield a été effectuée à l'aide d'arduino IDE afin de réaliser un prototype rapide et tester si le shield est efficace et permet de répondre à nos besoins.

Nous nous basons sur l'exemple proposé par arduino IDE suivant :

```
#include <SPI.h>
#include <SD.h>

File myFile;

void setup() {
// Open serial communications and wait for port to open:
Serial.begin(9600);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB port only
}

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {
Serial.println("initialization failed!");
return;
}
Serial.println("initialization done.");
myFile = SD.open("test.txt", FILE_WRITE);

if (myFile) {
Serial.print("Writing to test.txt...");
myFile.println("testing 1, 2, 3.");
myFile.close();
Serial.println("done.");
} else {
Serial.println("error opening test.txt");
}

myFile = SD.open("test.txt");
if (myFile) {
Serial.println("test.txt:");
while (myFile.available()) {
Serial.write(myFile.read());
}
```

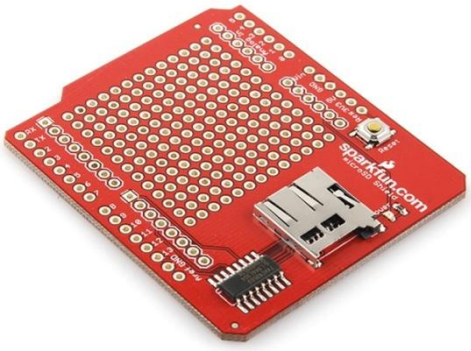
```
myFile.close();
} else {
Serial.println("error opening test.txt");
}
}

void loop() {}
```

Nous remarquons que nous écrivons facilement sur la carte SD grâce à la bibliothèque disponible. En ajoutant quelques boutons et en modifiant légèrement le code, nous arrivons à sauvegarder la durée de maintien du bouton ainsi que la fréquence d'actionnement.

Une fois le prototype fonctionnel, nous décidons d'écrire le code en C. Pour cela, nous utilisons en particulier deux bibliothèques fournies par Monsieur REDON : Sd2Card et spi. Après de multiples tentatives, nous arrivons toujours à l'erreur suivante : "SD_CARD_ERROR_CMD0". Cette dernière est liée à un timeout qui persiste malgré la modification de paramètres dépendants.

Monsieur REDON nous a fourni un nouveau *shield* uniquement SD cette fois-ci pour tester si le problème est matériel. En voici la photographie :



En recompilant le code sur le nouveau *shield*, nous remarquons qu'il n'y a pu d'erreur liée à l'initialisation de la carte SD. Suite à des expérimentations liées à la prise en main du code exemple donné, nous arrivons maintenant à écrire une information dans un bloc est à relire cette information plus tard.

Voici un extrait lu depuis l'utilitaire minicom lors de la lecture de la carte SD
status de la carte sd = 1
status=0
erreur=0
type=3
taille=7744512
Lecture
statut=0
bloc[0]=1 bloc[1]=0 bloc[2]=aa bloc[3]=bb bloc[4]=cc bloc[5]=0
statut=0
bloc[0]=dd bloc[1]=ee bloc[2]=ff bloc[3]=0 bloc[4]=0 bloc[5]=0
Le shield fonctionne comme désiré et nous pourrons ainsi enregistrer des informations afin de les traiter plus tard.

iii. Le shield temps réel

Afin de compléter le suivi de l'animal, nous avons décidé d'utiliser un module temps réel ADA3296 (basé sur un DS1307). Ce module, une fois configuré et initialisé, pourra nous donner beaucoup d'informations sur l'alimentation de l'animal. Nous aurions pu utiliser les fonctions disponibles sur l'arduino, qui calculent le temps qui passe, mais cela aurait posé deux problèmes majeurs :

- L'utilisation gourmande du processeur pour calculer les durées (ce qui aurait fortement ralenti le fonctionnement du système entier de la gamelle).
- L'information est de moins en moins fiable sur la durée.

Comme pour le *shield* SD, nous avons au préalable utilisé Arduino IDE pour vérifier le bon fonctionnement du *shield* temps réel et vérifier qu'il nous fournit bien des valeurs cohérentes. Suite à l'expérimentation, nous notons que le *shield* nous retourne bien les informations requises pour l'étude de l'alimentation du chat. En effet, nous recevons des données de la forme : jour/mois/année/heure/minute/seconde.

Il est à noter que, dans notre cas, l'utilisation de la précision à la seconde près n'est pas utile.

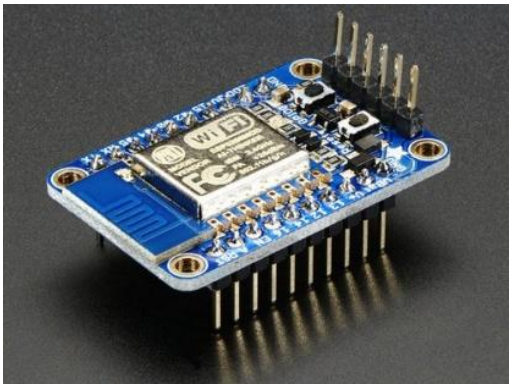
Finalement, le programme en C n'a pas eu le temps d'être réalisé. Seul un prototype sous arduino IDE l'est.

III. Connexion de l'objet avec l'application mobile

a. L'ESP8266

i. Présentation de l'ESP8266

Tout d'abord, l'ESP8266 est un circuit intégré à un microcontrôleur qui fait office de module WiFi. Il possède un total de 9 GPIO (3.3V) ce qui lui permet d'être utilisé indépendamment, sans même se connecter à un autre microcontrôleur. Pour notre projet, l'utilisation seule de ce module ne suffisait pas à cause du nombre de GPIO. De plus, nous souhaitons utiliser un shield carte SD et un shield temps réel, ce qui complexifie d'avantage le problème. C'est pourquoi, nous le connectons à un arduino afin de pouvoir profiter des différentes entrées/sorties de ce microcontrôleur. Voici une photographie du composant en question :



L'adresse IP de notre ESP8266 s'obtient en connectant notre composant à un réseau local. L'adresse attribuée varie donc en fonction du wifi auquel le module est connecté. Cette adresse est importante car grâce à elle nous serons capables d'établir un réseau.

Ensuite, la programmation en C du module wifi nous a pris beaucoup de temps pour finalement ne pas aboutir à quelque chose de fonctionnel. Le problème venait certainement du fait que nous devions implémenter le programme directement sur l'ESP8266. Nous avons donc vu avec nos encadrants afin de réaliser la configuration sur Arduino IDE. M. REDON nous a alors autorisés à réaliser uniquement le programme de l'ESP sur Arduino IDE. La tâche a été alors plus aisée puisqu'une bibliothèque propre à notre module wifi existe. De plus, il faut veiller à s'assurer que la version de l'IDE soit suffisamment récente afin de permettre l'ajout de *board*. En effet, il faut, grâce à un lien disponible sur le site d'arduino, ajouter l'ESP8266 en tant que board dans le logiciel afin que le composant puisse être reconnu avant de téléverser un programme. Ensuite, l'ESP doit être connectée en USB à l'ordinateur afin qu'elle soit visible sur son port propre. Ainsi, lors du téléversement du fichier, il faut sélectionner ce port dans la liste de ceux disponibles. (Beaucoup de soucis ont été rencontrés à ce propos, l'ESP n'étant pas toujours visible comme étant disponible sur arduino IDE). Un test basique consistant à faire clignoter une LED connectée à un GPIO de l'ESP a alors été réalisé. De plus, nous nous sommes connectés à un réseau wifi. Un simple « #include <ESP8266WiFi.h> » nous a permis de le réaliser grâce à la bibliothèque correspondante. La fonction *WiFi.begin* permet alors de se connecter facilement à un réseau local en passant en paramètres le nom du réseau sans fil (SSID) ainsi que son mot de passe.

Voici une visualisation des informations envoyées sur l'interface série (l'ESP fonctionne à 115200 baud) :

L'avantage d'être passé par Arduino IDE reste donc que l'implémentation du programme directement sur le module wifi ne posait alors plus de problème. Ainsi, nous pouvons programmer le reste de notre projet en C, en compilant et téléversant le fichier sur l'arduino sans perturber le fonctionnement du module WiFi.

ii. Création d'un réseau

La communication entre le module WiFi et l'application mobile est une partie très importante. Pour ce faire, un serveur TCP a été implémenté sur l'ESP. En utilisant la commande *Netcat*, nous avons pu vérifier qu'il était fonctionnel. En outre, l'idée était d'envoyer une requête permettant d'allumer une LED connectée à un GPIO de l'ESP et de l'éteindre par une autre. Le serveur réalisé, il fallait maintenant établir la communication avec l'application mobile. De ce côté-là, l'application agissait en temps que client. La création de socket était au programme. C'est alors que connaître l'adresse IP du module WiFi devient primordial. En effet, le constructeur de socket en Java utilisé prend en paramètres l'adresse IP du module de type *InetAddress* ainsi que le port de communication utilisé. (Ce port est spécifié dans le programme IDE du serveur TCP).

Il est à noter que pour réaliser une partie réseau sur une application mobile, il est préférable de créer un *thread* secondaire afin de rendre l'application la plus fluide possible. En l'occurrence, l'appui du bouton ON notamment, permet de modifier une chaîne de caractère et lance le thread d'envoi. Dans ce thread, l'application connecte le socket au module WiFI. Ensuite, le socket se charge d'envoyer la chaine de caractère à l'esp8266. En ce qui concerne l'envoi de la donnée, nous passons par l'objet *DataOutputStream* et sa méthode *writeBytes*. Nous déclarons d'abord un *OutputStream* pour y récupérer l'objet correspondant dans le socket. Suite à cela, nous convertissons l'objet *OutputStream* en objet *DataOutputStream* via le constructeur correspondant.

Ainsi, pour construire notre socket, nous le faisons comme suit :

```
InetAddress esp_ip = InetAddress.getByName("192.168.1.40");
socket = new Socket(esp_ip, SERVERPORT);
```

L'envoi des données s'effectue alors comme cela, la variable *text* étant le *string* à envoyer :

```
OutputStream data=socket.getOutputStream();
DataOutputStream d = new DataOutputStream(data);
d.writeBytes(text);
```

Voici un aperçu de ce qui est visualisable sur l'interface série de l'arduino IDE, les requêtes sont bien envoyées :

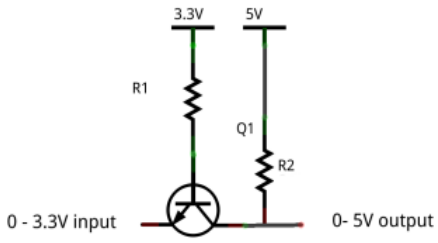
```
WiFi connected
Server started
192.168.1.24
new client
/gpio/0
Client disonnected
new client
/gpio/1
Client disonnected
new client
/gpio/0
Client disonnected
new client
/gpio/1
```

Le problème important qui a été rencontré pour réaliser cette connexion nous a fait perdre un mois sur la durée du projet. En effet, le *socket* ne se connectait jamais et ressortait des exceptions liées à la route et à l'adresse IP du module WiFi. Un mois s'est écoulé avant que nous nous rendions compte que le problème provenait du téléphone qui était utilisé en point d'accès internet. L'ESP8266 était connectée à ce WiFi et le téléphone lançant l'application mobile aussi. En réalité, la version

d'android du téléphone empêchait tout bonnement le *socket* de se créer et de connecter l'application au module wifi.

De plus, un de nos objectifs principaux était de pouvoir contrôler le servomoteur moteur depuis l'application mobile afin de renflouer la gamelle. Pour ce faire, nous souhaitons dans un premier temps passer par le port série, en connectant le RX de l'ESP au TX de l'arduino et inversement. Nous avons essayé de cette manière mais également en sollicitant un *SoftwareSerial*, objet disponible sous arduino IDE. Le problème avec cette deuxième méthode était de le réaliser sur l'arduino en langage C. Les résultats ne furent pas probants. Ainsi, la solution pour laquelle nous avons finalement opté était de relier un GPIO de l'ESP8266 à un PIN digital de l'arduino. Ainsi, étant capables d'allumer une LED par appui d'une touche sur l'application mobile, nous serions capables de faire changer l'état d'un PIN de l'arduino et ainsi en fonction de cela, faire tourner le moteur.

En mettant en place cette méthode, nous nous sommes rendu compte que le PIN de l'arduino ne réagissait étrangement pas à du 3.3V, envoyé par le module wifi. Nous avons donc mis en place un *high shifter* à l'aide d'un transistor afin de faire passer la tension d'entrée de 3.3V à 5V.



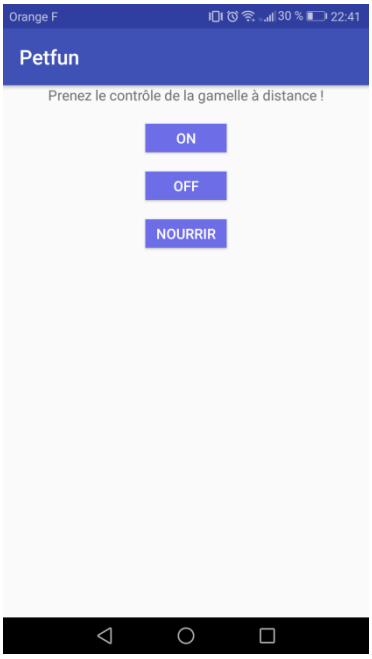
L'implantation de ce level-shifter a permis de faire tourner le servomoteur sans soucis. Il ne nous restait malheureusement plus suffisamment de temps pour revenir sur le problème de la transmission de données pures entre l'arduino et l'ESP8266. Nous pensons néanmoins que ce problème aurait pu être résolu grâce au level-shifter.

b. L'application mobile *Petfun*

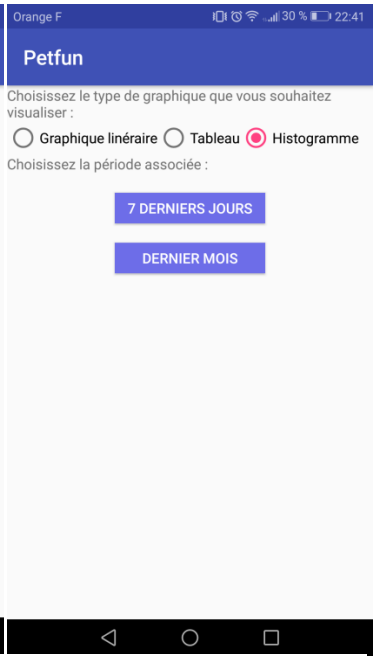
i. Interface

L'application a été réalisée en six activités. Sur le menu principal, quatre boutons apparaissent. Le premier ouvre l'activité liée au contrôle manuel de la gamelle. Le deuxième l'activité permettant le traitement des données de l'animal. La troisième ouvre l'activité de configuration de la gamelle (mode automatique). Enfin, la touche MAJ permet de solliciter le module WiFi afin de mettre à jour la base de données de l'application avec les valeurs stockées sur la carte SD présente sur l'arduino.

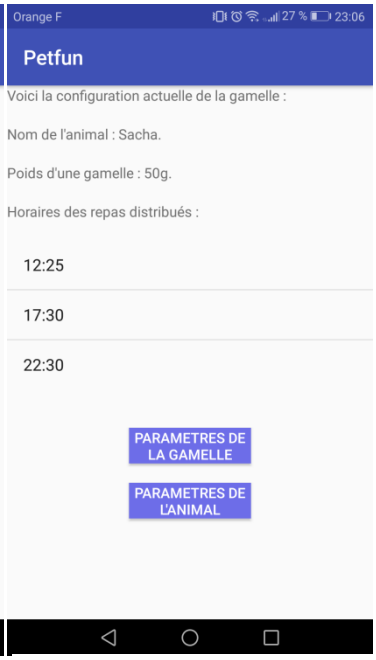
Index



Mode manuel



Activité statistiques



Mode configuration

Nous reviendrons un peu plus en détails sur ces parties par la suite.

ii. La base de données

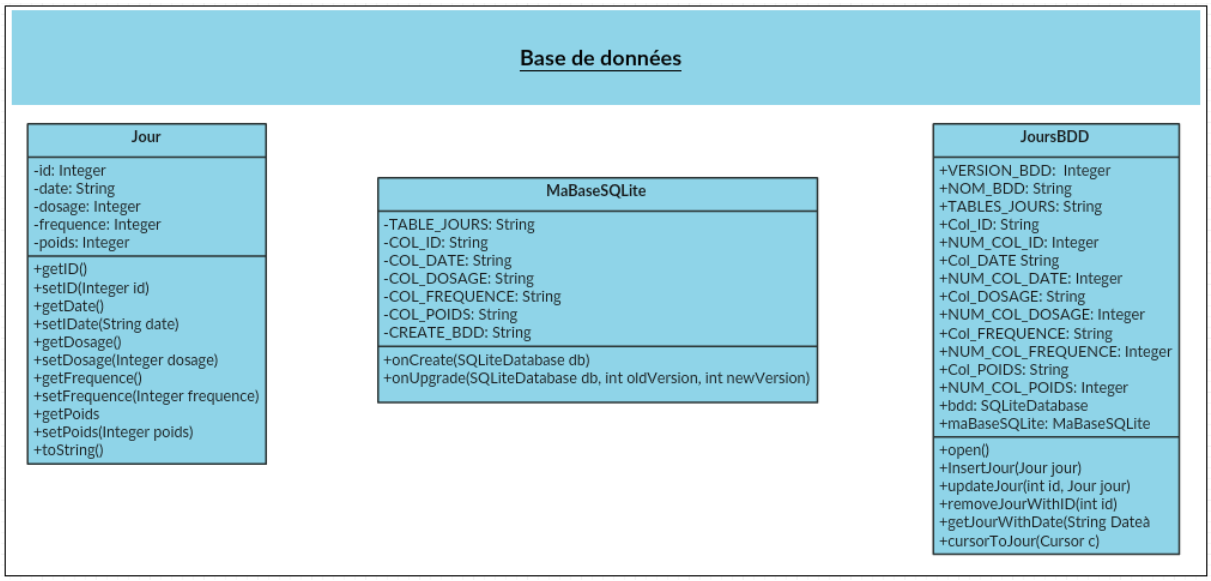
Nous sauvegardons déjà les données directement sur la gamelle grâce à la carte SD. Cependant, il nous a semblé nécessaire de garder en mémoire les statistiques de notre animal de compagnie directement sur l'application. Après avoir lu le livre « L'art du développement Android » de Grant Allen, prêté notre enseignant, deux possibilités nous sont apparues. La première était de manipuler des fichiers qui seraient générés par l'application afin de garder en mémoire ce que nous souhaitons. La seconde était de créer une base de données locale grâce à la base de données installée sur Android, *SQLite*.

SQLite utilise un dialecte de SQL afin de réaliser des requêtes de manipulations de données. Les « SELECT », « INSERT », « REMOVE », peuvent ainsi être utilisées. *SQLite* est particulièrement approprié pour l'utilisation android car son utilisation ne repose pas sur un modèle client-serveur (comme *mysql* ou *PostgreSQL* que nous connaissons). Ainsi, l'intégralité de la base de données est stockée dans un fichier unique.

La base de données que nous avons implémentée permet de répertorier tous les jours enregistrés par la gamelle afin de pouvoir réaliser des traitements statistiques derrière. La date a été stockée en chaine de caractère car le constructeur de la forme *Date(int year, int month, int day)* est obsolète pour la version de l'API que nous utilisions. Le deuxième constructeur est de la forme *Date(long Date)* et il faut fournir à cette méthode un nombre de millisecondes qu'elle va convertir au format Date de Java. Cela nous semblait peu pratique.
Voici un petit tableau illustrant ce que nous souhaitons stocker dans la base de données :

Date	Dosage (en g, selon le réglage de la gamelle)	Fréquence	Poids en kg de l'animal (facultatif)
01-04.17	30	2	50 (0 si non voulu)
02-04-17	50	1	50 (0 si non voulu)
…	…	…	…
…	…	…	…

Trois classes ont été crées : Jour.java, JoursBDD.java et MaBaseSQLite.java.
Voici un schéma UML résumant ces trois classes :



Enfin, nous avons dû modifier légèrement les classes liées à la base de données. Notre base de données étant créée et fonctionnelle, nous avions besoin de l'avoir à disposition n'importe où dans l'application. Or, passer un objet d'une page à une autre est simple pour des objets peu complexes. Cependant, il était plus difficile de passer notre base de données avec les méthodes traditionnelles utilisées auparavant. Nous avons donc utilisé un singleton. Le singleton est une instance unique d'une classe. Cela permet à la fois de récupérer facilement cette instance partout dans l'application mais cela permet également de converser une unicité de cette dernière, ce qui est important pour une base de données. Ainsi, nous avons modifié les classes décrites précédemment afin d'implémenter le singleton.

Deux variables ont été ajoutées à la classe joursBDD :

```
private static Context context;
private static JoursBDDSingleton mJoursBDDSingleton; //instance unique
```

Deux méthodes sont également concernées dans ce changement :

```
//Méthode de restauration de contexte
public JoursBDD(Context context)
{
    this.context = context;
    //creation de la base de donnees
    maBaseSQLite = new MaBaseSQLite(context, NOM_BDD, null, VERSION_BDD);
}

//Methode de recuperation de l'instance de la BDD
public static synchronized JoursBD getInstance(Context context){
    //Création de la base de données et de sa table
    if(mJoursBDD==null) {
        //CreateJoursBDD(context);
        mJoursBDD = new JoursBDD(context);
    }
    return mJoursBDD;
}
```

Ainsi, juste un réalisant une récupération de l'instance grâce à l'appel de la méthode *getInstance* : JoursBDD.*getInstance*(**this**); , il est aisé de le faire dans une nouvelle activité.

iii. Les graphiques

Une partie « *statistiques* » a été ajoutée à l'application mobile. Elle a pour but l'étude de l'alimentation de l'animal grâce à l'utilisation de graphiques. Afin de rendre la chose plus visuelle, l'utilisateur peut sélectionner soit un histogramme, soit un graphique linéaire, soit un tableau de données grâce à des radio boutons. Ensuite, l'utilisateur sélectionne soit les 7 derniers jours soit le dernier mois.

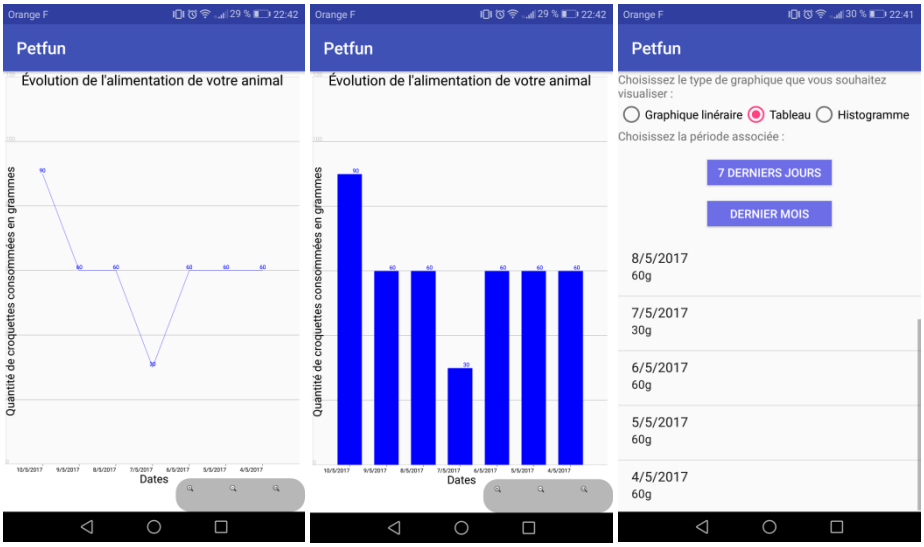
Que l'utilisateur ait sélectionné les 7 derniers jours ou le dernier mois, le principe reste le même. L'application va créer trois tableaux. L'un contenant les x dernières dates, en récupérant au préalable la date du jour grâce à l'objet *calendar*. L'autre contenant le nombre de gamelles ayant été vidées pour les x derniers jours. 'x' représentant 7 ou 30. Afin de récupérer les dates souhaitées dans le tableau correspondant, nous avons utilisé les id de chaque entité stockée dans la base. En effet, si la date 04/05/2017 a pour id 28, la date 03/05/2017 aura la date 27. Ainsi, il est plus simple de récupérer chaque donnée. Pour ce faire, nous avons simplement rajouté une méthode de récupération de l'id dans la classe JoursBDD.java.

```
public Jour getJourWithID(int id){
    Cursor c = bdd.query(TABLE_JOURS, new String[] {COL_ID , COL_DATE,
COL_DOSAGE, COL_FREQUENCE, COL_POIDS}, COL_ID + " LIKE \"" + id + "\"",
null, null, null, null);
    return cursorToJour(c);
}
```

Il est à noter que si l'application n'est pas à jour, un *toast* signale à l'utilisateur qu'il doit faire une mise à jour de sa base de données. L'affichage du tableau s'effectue grâce à un objet *ListView.* Pour lui affecter le tableau à afficher, il faut utiliser la méthode l'objet *ListView* qui est *lv.setAdapter(SimpleAdapter adapter).* La création de *l'adapter* repose sur l'utilisation d'une *hashmap* contenant les dates et le nombre de gamelles vidées. Un ScrollView permet de visualiser de longs tableaux.

Afin de réaliser l'affichage de graphiques, nous avons utilisé une bibliothèque appelée *achartengine*. Cette bibliothèque permet la personnalisation de graphiques et l'affichage sous forme d'histogramme ou encore de graphiques linéaires des données graphiques. Cette bibliothèque facilite la dynamisation de l'activité. Nous restant peu de temps dans le projet, nous avons décidé d'exploiter cette ressource afin de réaliser une page propre et personnalisée. Nous pouvons utiliser cette bibliothèque en l'ajoutant dans les fichiers *gradle* du projet.

Voici des les images relatives à cette activité :



Graphique linéaire *Histogramme* *Tableau et ScrollView*

Nous pouvons d'or et déjà vous évoquer les améliorations auxquelles nous avions songés.
Limites et améliorations :

L'idée d'avoir créée un bouton permettant d'afficher un graphique de la dernière semaine avait pour but une amélioration par la suite. En effet, l'objectif est de pouvoir, une fois le graphique affiché, passer d'une semaine à une autre. De même pour les mois, nous souhaiterions pouvoir, à partir du dernier mois, afficher les précédents en *slidant* le graphique ou en appuyant sur une touche.

iv. La configuration

Une page de contrôle automatique de la gamelle a été ajoutée à l'application mobile. Elle a pour but d'enregistrer des horaires auxquelles des croquettes seront déversées dans la gamelle, le poids contenu dans la gamelle, le nom de l'animal ou encore sa photographie. Le poids contenu dans la gamelle doit être notifié car cette dernière est ajustable grâce à des rails. Ainsi, l'utilisateur peut doser la quantité qu'il souhaite. Il devra le notifier dans l'application.

Dans l'activité de contrôle *automatique*, il s'agit en fait de configurer des paramètres. Sur cette page, les paramètres actuels de la gamelle sont affichés à l'écran. L'utilisateur peut dès lors, via des boutons, sélectionner deux activités : une pour changer les paramètres propres à la gamelle et l'autre pour changer ceux propres à l'animal.

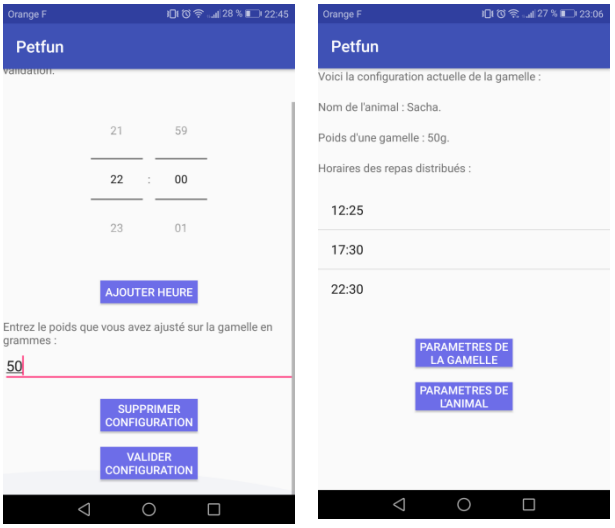
1. Paramètres de la gamelle

Dans les paramètres de la gamelle, deux choses sont configurables : les heures de repas de l'animal et la quantité de croquettes ajustée sur la gamelle.

Afin de rentrer les heures de repas, le widget *TimePicker* a été utilisé et configuré en format 24 heures. Il permet de rentrer des heures de manière très intuitive comme lorsque nous configurons nos réveils habituellement sur le smartphone. L'appui de la touche « AJOUT HEURE » permet l'ajout de l'heure saisi sur le *TimePicker* dans les paramètres de l'application. Nous allons revenir sur cette notion un peu plus bas. De plus, afin de saisir le poids de la gamelle, un simple *EditText* permet de le faire. Si l'utilisateur ne saisi pas un nombre, un message s'affiche à l'écran le prévenant du problème de format.

L'appui de la touche « VALIDER CONFIGURATION » permet la sauvegarde de la saisie et le retour à la première activité.

L'appui de la touche « SUPPRIMER CONFIGURATION » permet la suppression des heures de repas enregistrées dans l'application.



2. Paramètres de l'animal

Dans les paramètres de l'animal, le nom ainsi que la photographie de ce dernier peuvent être ajoutés. Le nom est simplement enregistré grâce à un *EditText*. L'avatar peut être cherché par l'utilisateur dans ses fichiers de téléphone. Les permissions adéquates ont ainsi dues être renseignées dans le fichier *manifest*. Un *bitmap* se crée alors et est stocké sous forme de *Singleton* afin de pouvoir être affiché en première page de l'application.

Maintenant, la question d'enregistrer tous ses paramètres s'est posée. En effet, une fois l'application fermée, tous les objets et toutes les variables créées dans l'application sont désallouées. Les paramètres rentrés par l'utilisateur sont alors perdus.

Plusieurs solutions s'offraient à nous. Nous pouvions écrire dans un fichier interne, écrire dans un fichier externe, modifier les préférences de l'application ou encore de créer une nouvelle base de données SQLite. Nous avons fait le choix de modifier les préférences de l'application car il était pour nous suffisant d'utiliser cette méthode plutôt que de solliciter un fichier. Cette méthode est très fluide si de simples paramètres sont à enregistrer, ce qui est notre cas. Ces préférences renseignent des paires clé-valeur de données.

Pour ce faire, nous devons créer un objet *SharePreferences* via la méthode *getSharedPreferences* en lui attribuant un nom unique et en privatisant son accès.

```
final SharedPreferences prefs = getSharedPreferences("Configuration", Context.MODE_PRIVATE);
```

Dès lors, nous pouvons restaurer cet objet dans n'importe quelle activité, exactement comme un *Singleton*.
Pour créer ou modifier des préférences, nous devons appeler la méthode *edit()* de l'objet *SharedPreferences*. Une fois la valeur modifiée, il est impératif de faire appel à la méthode *apply()* ou *commit()* de l'objet afin de sauvegarder les changements apportés.

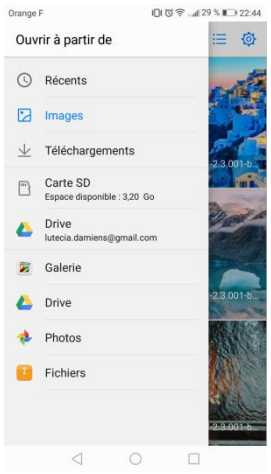
```
final SharedPreferences.Editor editor = prefs.edit();
```

Ainsi, les horaires sont stockées avec une clé de la forme « horairek », k représentant la numérotation des différentes heures saisies par l'utilisateur. Une variable « Taille » fait également partie des préférences afin de pouvoir savoir à tout moment le nombre de repas programmés par l'utilisateur. Le nom de l'animal est stocké avec la clé « Name » et le poids avec la clé « Poids ». . Il est alors aisé de retrouvé les valeurs correspondantes dans les préférences.

Il est à noter qu'il a fallu tenir compte de la possible mise à jour des préférences et au possible ajout d'horaire qui ne doit pas enduire de suppression totale de la liste correspondante.

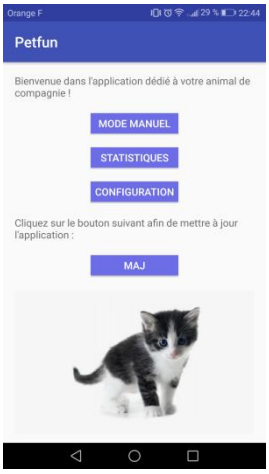
Choix du nom de l'animal

Validation du nom



Recherche de la photo

Sélection de l'a photo



L'image apparait sur l'index en première page

Conclusion

Ce projet s'inscrit dans le cadre de notre quatrième année d'IMA (Informatique, Microélectronique et Automatique) à Polytech Lille.

L'objectif était de créer une gamelle connectée originale. Un travail de recherche a été effectué en amont afin de connaître le marché des gamelles connectées aujourd'hui. Nous nous devions de créer un objet innovant. C'est pourquoi l'état de l'art était primordial.

Ce projet nous a permis la mise en pratique de nos connaissances et l'approfondissement de ces dernières (Java, socket, thread, arduino (avr), SQL). De plus, nous avons travaillé la plupart du temps en autonomie, ce qui nous a formés.

Ce projet nous a apporté beaucoup car nous avons fait de réels choix technologiques et matériels. De plus, nous avons pu travailler sur toutes les parties du projet : mécanique et logicielle. Ce travail a été intéressant car, dans le cadre d'un projet entreprise, plusieurs ingénieurs se divisent les tâches, ne permettant pas forcément l'investissement de chacun dans chaque partie.