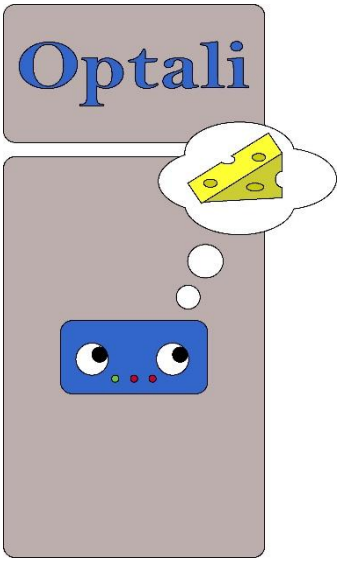
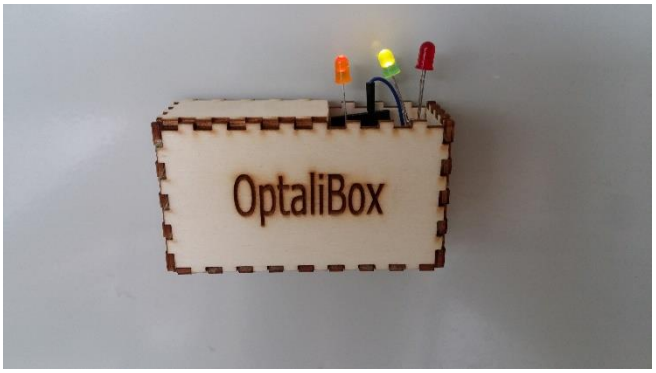
Aide à l'anti-gaspillage alimentaire Optali

LEFEVRE FRANCOIS / HUET ALEXANDRE





Année 2016-2017

Projet IMA4

Tuteurs de projet : Xavier REDON
Alexandre BOE
Thomas VANTROYS

Remerciements

Premièrement, nous souhaitons remercier l'équipe pédagogique de Polytech Lille pour nous avoir enseigné les connaissances pour mener à bien ce projet de 4ème année au sein du département Informatique, Micro-électronique et Automatique.

Nous tenons aussi à remercier nos tuteurs Mr Xavier Redon, Thomas Vantroys et Alexandre Boé pour leur accompagnement dans l'accomplissement du projet.

Merci à Thierry Flamen pour ses conseils avisés en électronique et dans la conception de notre carte électronique.

Nous souhaitons remercier aussi le Fabricarium de Polytech Lille pour nous avons confié ses équipements pour la réalisation de l'OptaliBox.

Pour finir, un remerciement particulier à Laurent Engels pour le temps consacré au tournage et à la réalisation du montage vidéo.

Table des matières

Introduction

Dans le cadre de notre formation Informatique, Micro-électronique et Automatique à Polytech'lille, nous avons réalisé un projet de 120h tout au long du semestre 8. Nous avons choisi de travailler sur l'aide à l'anti-gaspillage alimentaire car il nous permettait à la fois de travailler sur une application mobile Android native, mais aussi de concevoir une carte électronique et de mettre en place un serveur de base de données.

En France, nous jetons en moyenne 21% des aliments que nous achetons, ce qui représente une perte de 100 à 160 euros par an par habitant. La raison principale de ce gaspillage est la date limite de consommation associée à chaque produit, malgré qu'un produit puisse encore être comestible après péremption. Il serait donc judicieux d'aider un utilisateur à gérer les produits dont il dispose dans son frigo pour qu'il pense à les consommer avant qu'ils deviennent périmés.

Pour résumer tout le travail accompli, nous commencerons par présenter globalement notre projet en donnant l'objectif et le cahier des charges défini, les différents choix que nous avons effectué ainsi qu'un calendrier prévisionnel sur la répartition des tâches. Nous développerons ensuite les différentes parties du projet, à savoir le serveur Raspberry, l'application mobile Optali et la réalisation de l'OptaliBox. Nous terminerons par donner notre ressenti sur le projet, en évoquant les problèmes rencontrés mais également les améliorations possibles.

Tous les programmes et fichiers créés sur les différents logiciels que l'on utilise se retrouvent via notre dépôt GitHub à l'adresse suivante : https://github.com/FrancoisLefevre12/Optali.git ou dans une archive disponible sur notre page Wiki sur le site https://projets-ima.polytech-lille.net:40079/.

I) Présentation du projet

1) Objectif et cahier des charges du projet

Notre but est d'aider les utilisateurs d'une part à mémoriser les denrées périssables qu'ils disposent dans leur réserve alimentaire et d'autre part d'avertir l'utilisateur de divers problèmes. Ceci peut être une notification qu'un aliment va bientôt dépasser sa date de péremption, que la porte du frigidaire est restée ouverte ou encore que le nombre d'aliments dans la réserve devient critique et qu'il faudra de nouveau le remplir. Enfin, le système aidera l'utilisateur à trouver des recettes permettant de consommer les aliments en fin de vie.

Notre système sera composé de 3 éléments :

Premièrement, un smartphone appartenant aux utilisateurs. Ces derniers permettront de scanner grâce à son appareil photo, le code barre du produit ainsi que sa date de péremption. Il servira aussi d'interface utilisateur pour notre système. En effet, c'est par cet outil que l'utilisateur pourra lister tous les aliments qu'il possède dans sa réserve en ayant la possibilité de les trier sous différentes conditions, ou encore voir les recettes possibles pour consommer des aliments bientôt périmés.

Deuxièmement, un serveur, de type Raspberry, qui permettra de stocker la base de donnée contenant les aliments de la réserve. Ce serveur sera connecté par wifi.

Troisièmement, un micro-contrôleur (RFduino) qui interagira avec une LED pour indiquer la mise sous tension du contrôleur, une LED pour prévenir l'utilisateur qu'un aliment va bientôt passer sa date de péremption et une dernière LED pour prévenir l'utilisateur que son niveau de réserve est critique. Il y aura également un buzzer et un accéléromètre pour émettre un signal sonore quand la porte du frigo est restée trop longtemps ouverte. Ce dernier sera fixé au frigo par l'intermédiaire d'un dispositif adapté qui sera un système aimanté qui se pose sur le frigo.

Nous implémenterons donc une application mobile Android qui fera le lien entre les 2 autres systèmes.

Pour la partie électronique que nous aurons à réaliser, nous utiliserons une Breadboard pour faire des premiers tests puis nous créerons une carte PCB pour notre montage final.

Figure 1: Raspberry Pi 3

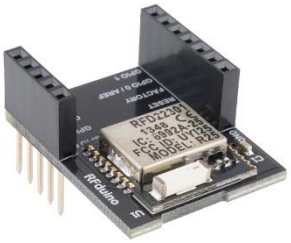


Figure 2 : contrôleur RFduino

2) Choix techniques et matériels

On utilisera une Raspberry Pi en guise de serveur pour pouvoir s'occuper du stockage de toutes les données dans notre projet puisque notre contrôleur sera incapable de les contenir, ce n'est d'ailleurs pas son but ici. En effet, même si la base de données commence à prendre de la place, nous n'aurons aucun souci à la stocker sur Raspberry. Ce n'est pas forcément le cas sur des microcontrôleurs où la mémoire reste limitée. De plus, étant connecté à un routeur, la Raspberry permet de diffuser les données de n'importe via le réseau Ethernet.

Notre smartphone fera donc le lien entre notre base de données et notre système embarqué par l'intermédiaire de l'application mobile. De plus, nous implémenterons un service web qui pourra gérer les différentes requêtes SQL entre l'application et notre base de données sur Raspberry.

Nous prendrons ici un RFduino comme micro-contrôleur pour notre OptaliBox puisque celui-ci nous permet d'ajouter des modules comme bon nous semble afin d'adapter le système aux besoins de notre application, comme le fait d'ajouter un module batterie pour que le contrôleur puisse être autonome. Il est également parfaitement compatible avec les applications Android. De plus, nous avons un Bluetooth directement intégré au RFduino pour s'assurer que les liaisons RFduino-application mobile se fassent correctement. On notera également que ce contrôleur consomme peu de courant et ne demande qu'une faible tension d'alimentation, critères importants pour ce genre de projet. C'est la raison principale de son utilisation pour être utilisé comme interface physique sur le frigo, en ajoutant également sa taille réduite comparée à la Raspberry Pi. Il possède aussi des pins programmables à souhait qui nous permettent d'interagir aisément avec des composants externes.

En fonction des objectifs que nous nous sommes fixés et des solutions proposées, nous avons pu définir une liste de matériel dont nous aurons besoin :

- Rfduino DIP board + USB shield (Mouser)
- Rfduino Dual AAA shield (Mouser)
- 2 piles de 1,5V AAA
- Raspberry Pi équipé du Wifi
- 1 buzzer (Mouser)
- 1 accéléromètre (Mouser)
- 3 LEDs de couleurs différentes (Mouser)
- Smartphone sous Android
- Aimants (Farnell)
- 3 Résistances CMS 330 Ohms (Mouser)

Concernant les outils de développement, nous utiliserons Android Studio pour développer notre application mobile principalement en Java. La programmation du RFduino se fera avec l'IDE Arduino. Le service web hébergé sur la Raspberry se fera en PHP et on utilisera MySql pour implanter la base de données. Pour la conception de la carte PCB, nous avons choisi le logiciel Altium Designer.

3) Calendrier prévisionnel

Suite à l'établissement du cahier des charges résumant le matériel nécessaire et les objectifs à atteindre, nous pouvons alors dresser une liste de tâches que nous aurons à effectuer :

- Configurer la Raspberry :
  - Installer et configurer le serveur
  - Créer le diagramme UML de notre future base de données
  - Créer la base sur le serveur
  - Configurer la connexion Wi-Fi
  - Implémenter le service web pour gérer nos requêtes SQL
- Monter et programmer le RFduino :
  - Réaliser une carte PCB pour connecter les différents modules au RFduino (accéléromètre, LEDs, buzzer).
  - Récupérer les consignes du smartphone grâce au Bluetooth
  - Programmer le comportement du RFduino en fonction des consignes
  - Envoi des alertes possibles (porte du frigo ouverte)
  - Construire le boitier final
- Implémenter l'application mobile Android :
  - Une première activité permettant de se connecter au RFduino et à la Raspberry
  - Scanner une date et un code barre
  - Ajout/suppression d'un aliment dans la base de données
  - Envoi de consignes au RFduino
  - Alerter l'utilisateur de différents problèmes avec des notifications
  - Rendre l'interface utilisateur ergonomique (lister les aliments, envoyer vers une page de recettes avec l'utilisation des aliments bientôt périmés...)

Après s'être donné une idée de ce qui devait être accompli, nous pouvons alors faire des prévisions sur la répartition de notre temps de travail sur toute la durée du projet, soit un total de 240 heures pour un binôme :

- Partie RFduino : 60h
- Partie Raspberry : 60h
- Partie développement mobile : 120h

II) Déroulement du projet

1) Le serveur Raspberry

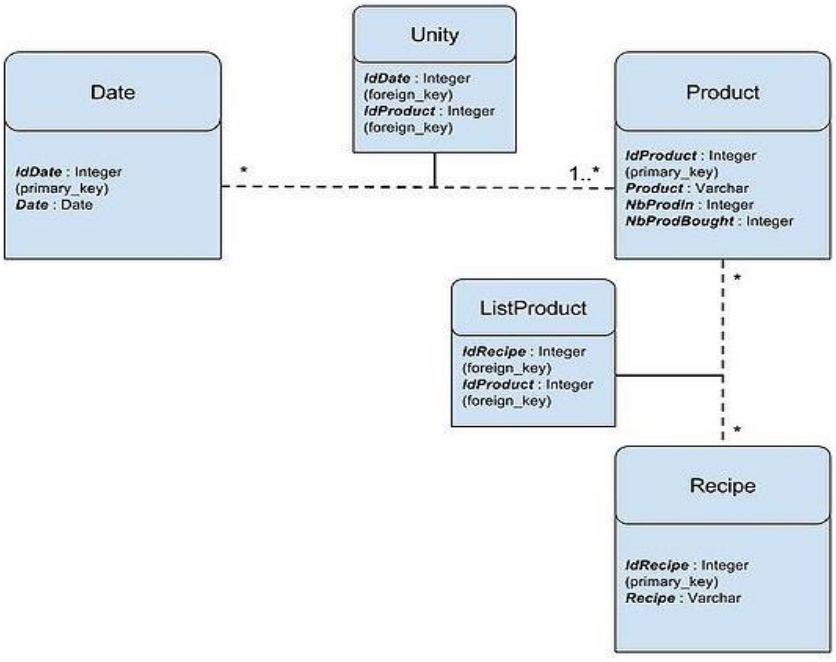
Dans notre projet, la Raspberry nous permet principalement de déporter notre base de données afin de ne pas surcharger notre application mobile. Cependant pour que les deux entités puissent communiquer, notre serveur doit être relié au réseau et accessible de n'importe où.

Nous avons donc dans un premier temps commencer par configurer la Raspberry via son port série en lui donnant une adresse IP fixe dans le fichier /etc/network/interfaces pour qu'elle soit connectée au réseau de l'école. Il fallait également utiliser le proxy de l'école ainsi que définir le serveur DNS associé à notre configuration réseau. Après avoir fait ceci, nous pouvons télécharger tous les paquets dont nous avons besoin afin de créer notre base de données et de pouvoir accéder à notre Raspberry en SSH.

Il ne restait donc plus qu'à implanter notre service web sur la Raspberry afin que notre base de données puisse être accessible par notre application mobile et faire les réglages nécessaires sur notre base de données. Cependant, il s'est avéré que la période où nous voulions commencer cette tâche tombait pendant une interruption pédagogique, ce qui nous empêchait de réutiliser la même configuration réseau pour notre Raspberry. Nous avons été contraints de modifier cette configuration.

Nous avons donc profité des vacances pour reconfigurer la Raspberry. Celle-ci est donc connectée en Ethernet sur notre réseau. Nous avons implanté Openssh-server pour pouvoir se connecter à distance à la Raspberry en SSH, Iptables pour sécuriser les ports non utilisés et libérer l'accès aux ports qui nous intéressent (port 22 pour SSH, 80 pour HTTP), Apache2 et PHP5 pour mettre en place notre service web et enfin Mysql-server pour accueillir la base de données. Nous avons ensuite créé un fichier de configuration Iptables, puis nous l'avons rendu exécutable et exécuté à chaque démarrage du système.

Vous trouverez ci-dessous le diagramme UML représentatif de notre base de données :



Les tables Product, Date et Unity servent à lister chaque produit du frigo de manière précise et efficace, sans effectuer de doublons dans la base de données. Les tables Recipe, ListProduct et Product servent quant à elles à lister les recettes ajoutées par l'utilisateur, l'objectif étant de savoir si elles sont faisables ou pas.

Pour chaque requête, une page web en PHP a été créée. Ainsi, on peut retrouver une page pour lister les produits, une page pour en ajouter un, une page pour lister les recettes et malheureusement, nous n'avons pas eu le temps de faire la page pour ajouter des recettes.

2) La conception de l'OptaliBox

L'OptaliBox n'est rien d'autre qu'un module complémentaire à l'application mobile qui vient se fixer sur le frigo auquel l'application est dédiée. Ce système est donc composé de notre RFduino contenant le programme permettant d'interagir avec les différents composants soudés sur notre carte PCB, qui vient se fixer directement sur le contrôleur. On rajoute également le shield pour pouvoir alimenter le RFduino avec deux piles AAA.

Pour que ce système puisse être fonctionnel, il fallait commencer par créer notre carte PCB pour pouvoir par la suite tester notre programme sur RFduino. Une fois que cela était fait, nous avons pu attaquer la conception de notre boitier qui contiendrait tout le système.

1. Conception de la carte PCB

Notre RFduino doit communiquer avec les différents composants électroniques. Afin que cela soit plus esthétique pour la modélisation finale, nous allons créer une carte PCB pour rassembler tous ces composants dessus.

Avant de représenter notre circuit général, nous allons commencer par créer une librairie rassemblant tous les composants que nous utiliserons par la suite, en particulier nos composants CMS (accéléromètre et résistances). Pour ce qui est des LEDs et du buzzer, qui sont traversants, nous pourrons utiliser des headers classiques.

Il a donc fallu créer le schematic ainsi que l'empreinte de nos composants CMS. Voici un exemple avec notre accéléromètre :

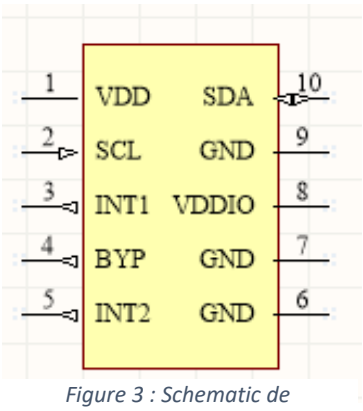


Figure 3 : Schematic de l'accéléromètre

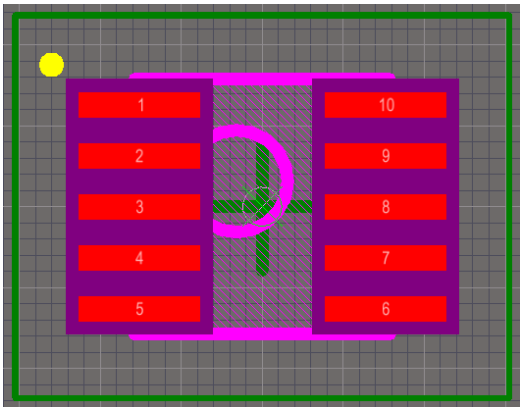


Figure 4: Empreinte PCB de l'accéléromètre

Après avoir fait ceci avec tous les composants qui nécessitent ce traitement particulier, nous pouvons maintenant construire notre schematic du montage entier et ensuite le fichier PCB afin de pouvoir graver notre circuit. Nous avons adapté la taille de notre PCB pour qu'il corresponde parfaitement à la taille du RFduino. Voici un aperçu du rendu final :

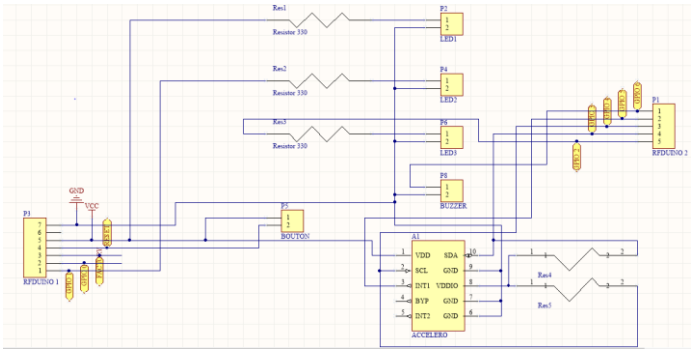


Figure 5: Schematic du circuit final

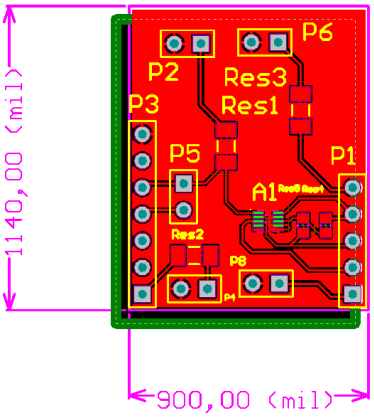


Figure 6: PCB du circuit final

On retrouve sur notre PCB final les headers, respectivement de 8 et 5 pins, de part et d'autre de la carte, qui représentent le RFduino, où nous souderons des broches pour qu'elles puissent être intégrées au RFduino. On note la présence de 5 autres headers de 2 pins, dont 3 pour des LEDs puis les 2 autres pour le buzzer et le bouton poussoir. Nous avons rajouté un bouton poussoir par rapport au circuit prévu pour pouvoir faire un Reset de notre RFduino lorsqu'on le souhaite. Ce qui reste sont les composants CMS avec plusieurs composants à 2 pattes qui sont toutes les résistances de notre montage, celles de protection pour les LEDs et les pull-ups pour l'accéléromètre. Le dernier composant à 10 pattes est donc bien l'accéléromètre.

Après avoir fait graver notre carte au service EEI et réalisé un bon nombre de soudures, autant au four qu'au fer à souder, nous obtenons notre carte PCB fonctionnelle :

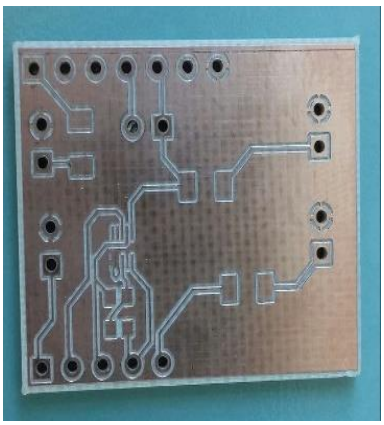


Figure 7: Carte PCB après gravure

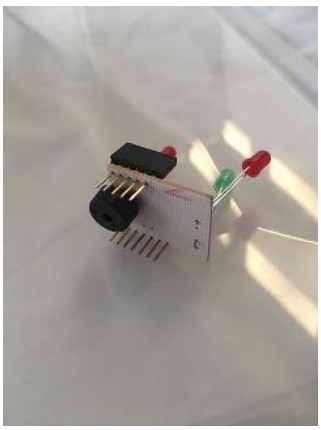


Figure 8: carte vue du dessous

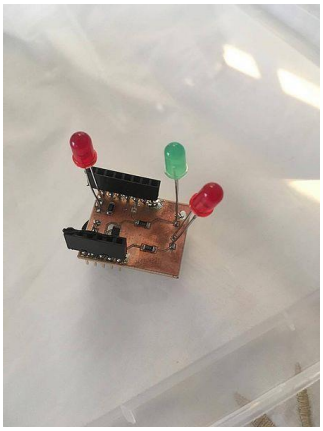


Figure 9: carte vue du dessus

2. Programmation du RFduino

Pour que notre contrôleur puisse répondre à nos attentes, c'est-à-dire interagir avec notre carte PCB avec les différents GPIOs et pouvoir communiquer par BLE (Bluetooth Low Energy) avec notre application mobile, il nous fallait donc le flasher avec le programme adéquat.

En utilisant principalement les librairies <RFduinoBLE.h> et <Wire.h> sous l'IDE Arduino, nous avons pu coder toutes les fonctions dont nous avions besoin.

La première nous permet d'utiliser d'activer le module BLE du RFduino et d'utiliser des fonctions d'envoi et de réception sans devoir définir de socket. Cela nous permet d'envoyer des chaînes de caractères pour notifier l'application mobile des différentes alertes, comme prévenir que le RFduino est bien connecté au téléphone. La fonction de réception, qui fonctionne par interruption, récupère quant à elle les alertes provenant de l'application.

La deuxième nous permet de définir une liaison i2C et de communiquer dessus en utilisant des fonctions simples. Ceci nous permet de récupérer les valeurs des axes X, Y et Z de notre accéléromètre et de voir si la porte du frigo est ouverte ou non, et d'activer le buzzer dans ce cas.

Pour le reste du programme, on ne fait qu'écrire des 0 ou des 1 sur nos GPIOs en fonction des alertes que l'on reçoit, pour allumer ou non des LEDs.

3. Réalisation du boitier

Nous avons conçu le boitier avec l'aide du Fabricarium de Polytech et de leur site. Nous avons extrait un fichier PDF contenant le patron de notre OptaliBox, nous l'avons retouché grâce à Inkscape, un logiciel de retouche photo, et nous avons utilisé la découpeuse laser et de la colle pour finaliser notre produit.

Après avoir fabriqué le boitier, nous avons ajouté des aimants sur celui-ci pour qu'il puisse être collé à une porte de frigo en contenant notre RFduino avec le PCB.

Figure 10: Fabrication du boitier

Figure 11: OptaliBox contenant le système complet

3) L'application mobile Optali

L'application mobile native Android possède 4 activités. Elle comprend la page d'accueil, l'activité d'ajout de produit, la liste des produits et la liste des recettes.

Premièrement, nous commençons avec la page d'accueil. Elle comprend la redirection vers les autres pages. Elle effectue une requête à la base de données et mémorise les alertes dans un singleton[1]. L'activité affiche les alertes de manière écrite et via des signaux. Il y a 4 signaux qui sont générés. Elle affiche un signal si le frigo est bientôt vide (c'est-à-dire moins de 30 aliments dans le frigo), si un aliment va périmer (3 jours avant d'atteindre la date de péremption), si la porte du frigo est restée ouverte et si l'application est connectée à l'OptaliBox. De plus, elle permet d'activer le Bluetooth et d'ainsi se connecter à l'OptaliBox via le bouton Scan. Une fois connectée, l'application envoie en continu à l'OptaliBox des booléens pour connaître l'état de chaque alerte, dont le traitement est expliqué précédemment. Pour l'application, il a fallu créer un socket Bluetooth permettant de gérer le protocole GATT, caractéristique du BLE. Pour cela, nous nous sommes inspirés d'un socket qui était déjà défini correctement.

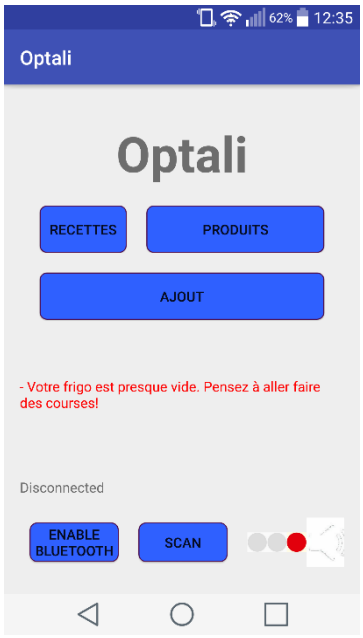


Figure 12 : Page d'accueil Optali

[1] Objet java à instance unique partagé entre toutes les activités

Deuxièmement, il y a l'activité d'ajout de produit. Elle possède trois modes d'ajout de produits. D'une part grâce à la saisie au clavier, d'autre part grâce à la complétion automatique. En effet, si un aliment a déjà été enregistré, il suffit de taper les deux premières lettres pour afficher le nom du produit et le rentrer en cliquant dessus. La dernière possibilité est d'effectuer une lecture des caractères de la boite avec la caméra du téléphone. Il suffit de cliquer sur le bouton « Scan », la caméra se lance, il faut ensuite capturer le nom de l'aliment, recadrer pour n'avoir que le nom de l'aliment et valider. Il en est de même pour l'insertion de la date. Quand la saisie est terminée, il faut ensuite taper sur : « Envoyer ».

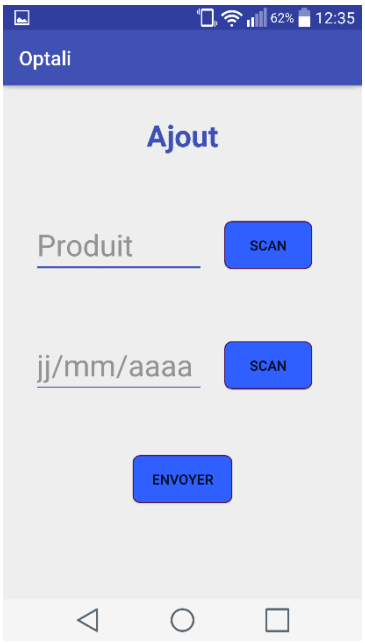


Figure 3 : Activité d'ajout de produits

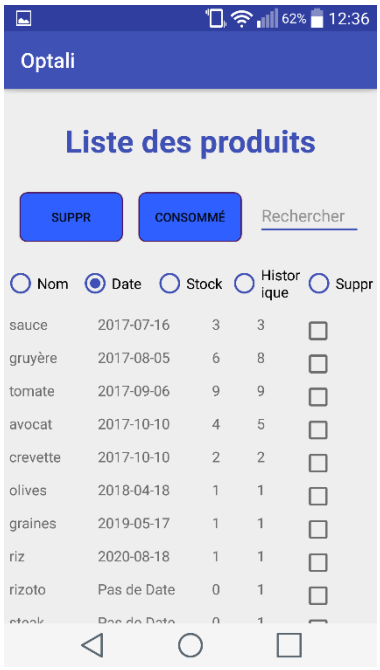


Figure 2 : Liste des produits

Troisièmement, nous avons la liste des produits. Elle trie par défaut les aliments dans la réserve alimentaire par leur date de péremption. Ainsi, on peut voir du premier coup d'œil lequel des aliments que l'on possède est préférable de manger en premier. On peut aussi grâce aux boutons radio les trier par nom, par quantité dans le frigo (Stock), par nombre d'ajout depuis le démarrage du dispositif (Historique). Nous pouvons aussi les sélectionner, soit pour signaler qu'on en a consommé un exemplaire, soit pour effacer une erreur de saisie par exemple. Elle dispose aussi d'une fonction de recherche, pour rechercher les aliments commençant par les lettres dans l'EditText.

Finalement, nous retrouvons la liste des recettes. Cette activité n'était pas obligatoire car elle n'était pas dans notre cahier des charges. Cependant nous souhaitions accentuer l'utilité du dispositif en faisant gagner encore plus de temps à l'utilisateur. Celui-ci pourrait ajouter lui-même ses propres recettes en fonction de ce qu'il a dans son frigo et l'application ferait le lien avec la base de données. Ainsi nous connaitrions les recettes réalisables. L'activité d'ajout n'existe pas encore mais les recettes peuvent être ajoutées via la base de données. En cliquant sur une recette, une liste apparait qui correspond aux aliments correspondants à la recette.

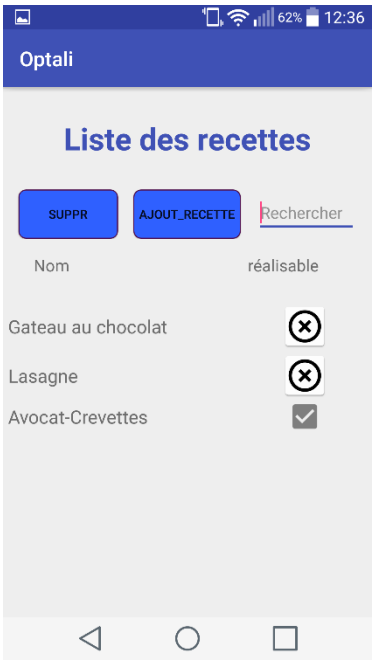


Figure 4 : Activité de liste des recettes

Les librairies qui ont été utilisées sont Zxing pour la détection de code-barres (qui a été implémentée mais par manque d'efficacité, remplacée par la lecture de caractères), Tesseract pour la lecture de caractères et Volley pour communiquer avec le serveur web.

III) Analyses personnelles

1) Difficultés rencontrés

Ce projet a été pour nous l'occasion de développer une application mobile pour la première fois. Il nous a fallu un certain temps pour prendre en main la technologie ainsi que les différents langages de programmation utilisés mais cela est un effort que nous n'aurons plus à faire.

Malgré que nous ayons déjà travaillé sous Altium un certain nombre de fois, nous avons mis un temps conséquent pour créer une carte PCB en utilisant ses propres composants. En effet, nous avions obligation de créer chacun de nos composants pour faire notre propre bibliothèque de composants, sauf pour les composants traversants qui sont déjà définis dans les bibliothèques classiques. A la fin de ce projet, il nous paraît maintenant plus facile de jongler entre les datasheets de nos composants et de trouver rapidement les informations dont nous avons besoin.

En partie à cause des raisons précédentes, nous avons perdu du temps sur la création de notre carte PCB, qui était pourtant relativement simple. A chaque fois que nous étions sur le point d'envoyer nos fichiers pour graver notre carte, nous avions toujours des légers détails à rectifier, ce qui retardait souvent l'échéance.

Le fait d'avoir choisi nous-mêmes tous les composants dont nous avions besoin nous a appris à vérifier différentes caractéristiques importantes pour un composant. Effectivement, au départ du projet nous n'avons pas porté une grande attention à la taille de nos composants CMS. Nous nous sommes rendus compte après coup que nous avions opté pour des composants très petits. La difficulté pour les souder est alors particulièrement augmentée. Cela restait aisé pour les résistances, cependant nous ne savons toujours pas si nous avons soudé correctement notre accéléromètre, tellement celui-ci est minuscule. Si c'était à refaire, nous prendrons donc un accéléromètre plus grand quitte à payer un peu plus cher le composant.

Sur notre carte PCB finale, il nous manque le bouton poussoir par lequel nous pouvions faire un reset de notre RFduino. En voulant le souder, nous avons malencontreusement arraché la piste qui reliait l'alimentation au bouton poussoir. Cela nous empêchait du coup d'allumer notre LED indiquant la mise sous tension du contrôleur et d'alimenter notre accéléromètre. Nous avons du coup directement soudé un fil de la broche d'alimentation vers la LED. Pour le bouton poussoir et l'accéléromètre, nous avons décidé de les laisser de côté.

Concernant la LED qui s'allume lorsqu'un produit est bientôt périmé, nous ne comprenions pas dans un premier temps pourquoi elle restait allumée malgré qu'elle soit bien soudée et que le programme était correct. Nous avons donc changé la LED en pensant que cela réglerait le problème, mais ce n'était pas le cas. Après quelques recherches, nous avons compris qu'une liaison série utilise de base les GPIOs 0 et 1 pour les broches RX et TX. Or la LED en question est connectée au GPIO 1, ce qui veut dire que nous avions un conflit sur cette broche. Nous avons juste changé les GPIOs de la liaison série sur des GPIOs inutilisés pour régler ce problème.

L'établissement de la communication BLE entre le RFduino et l'application nous a posé quelques soucis, surtout côté application. Initialement, nous voulions utiliser les broches RX et TX pour faire une communication Bluetooth simple avant de nous rendre compte que cela n'était pas compatible avec le BLE, qui utilise un protocole bien spécifique. Nous sommes donc repartis de zéro pour créer un socket en Java permettant de gérer le protocole GATT associé au BLE et ensuite l'implémenter dans notre activité principale. La compréhension de ce mécanisme fut assez difficile.

Cela n'a pas non plus été évident de trouver les technologies qui sont efficaces face à nos besoins. En effet, nous pensions l'utilisation de la lecture de code-barres efficace mais la caméra mettait du temps à effectuer la mise au point et il fallait en plus avoir une connexion internet pour interroger une base de données qui était, pour la majorité d'entre elles, payante ou incomplète.

2) Améliorations possibles

L'ajout d'aliment est fonctionnel. On peut lister les aliments du frigo, en ajouter et en consommer autant que l'on souhaite. Par contre, l'ajout de recette n'est pas encore implémenté. On peut en ajouter via la base de données SQL mais un utilisateur lambda ne serait pas capable de le faire. Il faudrait ajouter une activité qui sert juste d'ajout de recette.

Le point négatif au vu de notre cahier des charges est que nous n'avons pas réussi à faire fonctionner notre accéléromètre. Nous avons été incapables de déterminer si le problème était électrique ou venait du programme. Nous perdons donc une fonctionnalité de notre OptaliBox qui était de buzzer lorsque la porte du frigo restait ouverte. Il serait donc judicieux de trouver un accéléromètre plus grand pour le souder plus facilement et ainsi ajouter cette fonctionnalité qui peut s'avérer précieuse.

Même si nous avons finalement réussi à concevoir un système permettant d'englober notre RFduino et le PCB dans une boite pour le fixer au frigo, le design de celui-ci pourrait être à revoir dans le cas où il faudrait commercialiser le produit par exemple. En effet, nous avons profité des services du Fabricarium de l'école pour avoir quelque chose de suffisant. Il serait possible de travailler plus profondément l'aspect visuel de notre OptaliBox.

Une fois le programme du RFduino terminé, nous avons pu remarquer que nous n'utilisions qu'une infime partie de la mémoire du contrôleur. Si l'on devait réfléchir dans une optique commerciale, il pourrait être judicieux de remplacer le RFduino par un micro-contrôleur avec moins de mémoire qui sera donc moins coûteux, tout en conservant le module Bluetooth et les broches programmables à souhait. On peut se poser les mêmes questions concernant notre Raspberry Pi, qui est un micro-ordinateur bien puissant pour stocker seulement une base de données et gérer les requêtes SQL.

Conclusion

Dans ce compte-rendu de projet, nous avons commencé par présenter de manière générale notre projet en définissant l'objectif et le cahier des charges, les différents choix et stratégies que nous avons adopté ainsi qu'un calendrier prévisionnel sur la répartition des tâches. Nous avons ensuite évoqué les différentes composantes de notre projet, à savoir le serveur Raspberry, l'application mobile Optali et la réalisation de l'OptaliBox. Nous avons terminé par donner nos impressions sur le projet, en discutant des problèmes rencontrés mais également des améliorations possibles.

Nous pouvons dire que nous avons rempli tous les impératifs du cahier des charges initial, puisque nous avons juste l'accéléromètre qui n'est pas fonctionnel. Notre application mobile est bien capable de communiquer avec le RFduino ainsi que la Raspberry Pi pour d'une part échanger des données via Bluetooth et d'autre part accéder à notre serveur de base de données par PHP.

Néanmoins, nous savons que notre projet possède des limites et qu'on pourrait y apporter des améliorations. On pourrait par exemple avoir un design plus poussé ou travailler sur les conditions des différentes alertes pour rendre les notifications optimales et adaptées au contenu de la réserve alimentaire.

Pour remplir globalement notre cahier des charges dans les délais imposés, nous avons dû réaliser une répartition la plus efficace qui soit de notre travail. Nous avons appris à travailler en binôme sur le même projet durant tout un semestre et nous sommes satisfaits d'avoir abouti sur un système fonctionnel avec un prototype présentable.

Avec l'effervescence des objets connectés de nos jours, ce n'est bien évidemment pas une révolution que de penser au frigo connecté. Cependant, ceci étant un business encore nouveau, le prix moyen pour un frigo connecté est plutôt important. Il pourrait donc être judicieux de développer un outil semblable avec beaucoup moins de fonctionnalités tout en gardant l'aspect essentiel à nos yeux, qui est la gestion du stock alimentaire pour éviter le gaspillage. Pour les personnes souhaitant réaliser un suivi de leur réserve alimentaire, cela serait possible à moindre coût.

HUET Alexandre

LEFEVRE François