Projet TZ20
Boîtier de contrôle de présence des étudiants

Rapport final

<u>Etudiants :</u> Valentin Mercy / Thomas Duvinage

<u>Enseignant suiveur :</u> Alexis Flesch

utbm
université de technologie
Belfort-Montbéliard

Table des matières

Introduction

Ce rapport vient faire suite au rapport intermédiaire rendu au milieu du semestre. Il présentera l'ensemble des travaux réalisés par Thomas Duvinage et Valentin Mercy dans le cadre de l'unité de valeur TZ20 intitulée « Travaux de laboratoire », au printemps 2020, pour le projet de conception et fabrication d'un boîtier de contrôle de présence des étudiants de l'Université de Technologie de Belfort-Montbéliard suivi par M. Alexis Flesch.

Ce rapport est accompagné des documents suivants :

- La présentation utilisée pour la soutenance finale (PDF)
- Le mode d'emploi du boîtier (PDF)
- Le schéma électronique du boîtier (PDF)
- Un modèle de liste des étudiants à importer dans le boîtier (CSV)

Description du projet

Chaque année, plusieurs centaines d'étudiants rentrent à l'Université de Technologie de Belfort-Montbéliard. Un des passages obligatoires pour tous les étudiants de première année est la formation sécurité (il s'agit d'un exemple de ce qui sera dorénavant désigné par le terme « événement ») avant le premier départ en stage. Pour s'assurer que tous les étudiants ont participé à cette formation, l'UTBM a mis en place un système d'émargement informatisé.

Ce système actuel est constitué d'un lecteur RFID[1] USB qui permet aux étudiants de badger avec leur carte étudiante. Celui-ci est relié à un ordinateur portable qui enregistre l'UID (*Unique Identifier*) de chaque carte dans un fichier CSV[2].

Après chaque événement, la secrétaire du Tronc Commun UTBM et/ou ses collègues (qui seront dorénavant désignés par le terme « utilisateur ») respectent une certaine procédure :

1) Emport de l'équipement nécessaire (au moins deux appareils : le PC portable (lourd), le lecteur de cartes, et éventuellement un chargeur)
2) Sur le lieu de l'événement : les étudiants sont contraints de rentrer dans l'amphithéâtre un par un, car ils doivent tous se présenter au lecteur de cartes. Un programme exécuté sur le PC portable enregistre un à un les UID des cartes étudiantes. L'arrêt de l'exécution de ce programme met définitivement fin au contrôle de présence, et le matériel est rangé[3].
3) A la fin de l'événement : connexion de l'ordinateur à un réseau wifi et utilisation d'un second programme pour récupérer les logins[4] en fonction de l'UID de chaque carte.
4) Récupération auprès du service informatique (DSI) du fichier regroupant les informations d'identité des étudiants censés être présents lors de la formation. Avec ce fichier l'utilisateur extrait les noms et prénoms attachés au login. Cela génère deux fichiers : un premier constituant la liste des présents et l'autre la liste des absents.
5) Pour les étudiants qui n'ont pas badgé mais qui ont été signalés présents grâce à l'émargement manuscrit, ils sont retirés manuellement de la liste des absents.

[1] RFID = Radio-Frequency Identification (wikipedia.fr, 2020)

[2] CSV = Comma Separated Values. Il s'agit d'un fichier texte contenant des chaînes de caractères séparées par des virgules.

[3] Ainsi, les étudiants qui se présentent en retard sont listés de façon manuscrite, et retirés un à un de la liste des absents de retour au bureau.

[4] Les logins sont des chaînes de caractères courtes qui identifient chaque étudiant de l'UTBM de manière unique.

De plus, en fonction de l'endroit où il est situé, l'utilisateur n'a pas forcément la possibilité de brancher l'ordinateur à une prise électrique. Un autre problème vient de la fiabilité comme nous l'a fait remarquer Madame Cagnon (actuelle secrétaire du Tronc Commun) :

« Nous ne sommes pas sûrs que tous les étudiants aient bien été enregistrés (pas de signal sonore ou de diode lumineuse). Nous devons faire trop de manipulations pour extraire la liste des étudiants. »

Ainsi l'objectif de ce projet est d'automatiser ce processus afin de soulager les utilisateurs et les étudiants, en construisant un boîtier portable et léger intégrant tous les périphériques nécessaires au contrôle de présence. Cela permettra à terme de gagner en efficacité, en temps et de s'affranchir d'éventuelles erreurs de manipulations. Afin de supprimer les files d'attentes (dont la taille excède parfois les 200 étudiants), le boîtier devra pouvoir circuler d'étudiant en étudiant pour un contrôle ultérieur à leur installation dans l'amphithéâtre.

Pour cela le système doit avoir les caractéristiques suivantes :

- Être compact
  - Composé d'un seul boîtier portable
  - Pas de fils extérieurs
- Être autonome en énergie pour une durée d'au moins 1 heure
- Être rechargeable
- Être simple d'utilisation : limiter la manipulation du boitier au strict nécessaire
- Lire et générer des fichiers compatibles avec Excel
- Respecter le RGPD[5]
- Être capable d'émettre des signaux lumineux et sonore
- Limiter les interactions avec la DSI[6]
- Être inviolable et inextinguible par les étudiants dont on contrôle la présence
- Pouvoir communiquer avec une clé USB (en lecture et en écriture)
- Distinguer les « faux présents » (étudiants qui se présentent alors qu'ils ne sont pas invités à l'événement) des présents
- Être capable de fusionner les résultats d'un contrôle de présence à un autre contrôle réalisé sur un deuxième boîtier identique, dans le cas où l'on souhaiterait procéder à deux contrôles de présence pour un même événement afin, par exemple, de permettre une installation des étudiants plus rapide

Analyse fonctionnelle

Pour exprimer plus clairement le besoin du secrétariat, nous l'avons représenté sur un diagramme bête à cornes :

[5] RGPD = Règlement Général sur la Protection des Données

[6] DSI = Direction des Services Informatiques (à l'UTBM)

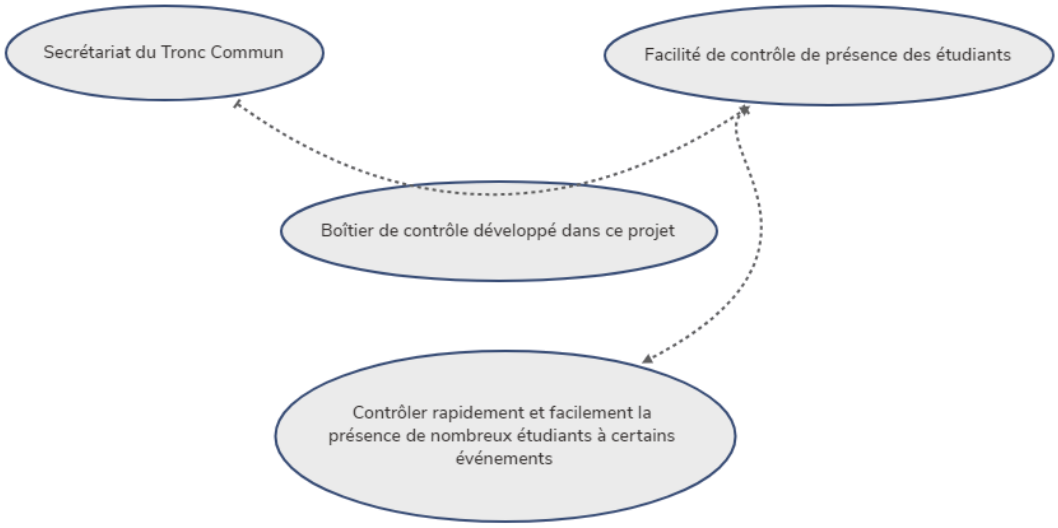


Figure 1 : Diagramme Bête à cornes

Nous avons ensuite établi un diagramme pieuvre :

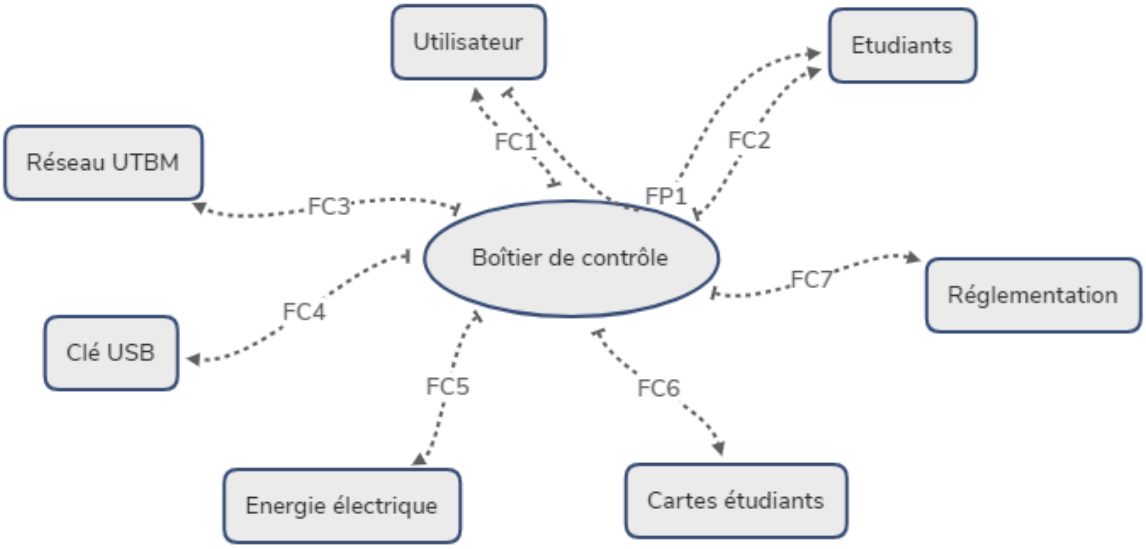


Figure 2 : Diagramme pieuvre du boîtier

Ce diagramme est lié aux fonctions suivantes :

- FP1 : Permettre à l'utilisateur de contrôler rapidement et facilement la présence de nombreux étudiants à certains événements
- FC1 : Proposer une interface ergonomique et facile à prendre en main
- FC2 : Être utilisable directement par les étudiants avec des restrictions d'accès
- FC3 : Être capable de communiquer avec le réseau informatique de l'UTBM pour identifier les étudiants
- FC4 : Être capable de lire la plupart des clés USB

- FC5 : Être autonome en énergie pendant la durée d'un contrôle (au moins une demi-heure)
- FC6 : Être capable de lire les cartes étudiants UTBM
- FC7 : Respecter la réglementation (respect de la vie privée)

Les composants nécessaires

Pour la réalisation de ce projet nous avons besoin de quelques composants électroniques classiques (la plupart étant disponibles préconditionnés en modules) ainsi que de certains éléments de fixation. Vous trouverez dans le tableau ci-dessous la description de l'ensemble des composants ainsi qu'un budget matériel estimatif pour le projet. Il faut également prévoir quelques consommables habituels, spécifiés en rouge dans le tableau.

Liste du matériel - Prix total HT : 86 €								
Désignation	**Catégorie**	**Référence**	**Caractéristiques**	**Qté**	**Exemple**			
					URL	**Prix HT**	**Fournisseur**	**Prix total HT**
Raspberry Pi	Eléctronique	3B+	Modèle 3B+ ou 4	1	Cliquer ici	36,63 €	Gotronic	36,63 €
Ventilateur	Eléctronique		30*30*10mm – 5V	1	Cliquer ici	8,29 €	Gotronic	8,29 €
Ecran LCD + interface I2C	Eléctronique		16x2 – avec module I2C	1	Cliquer ici	8,25 €	Gotronic	8,25 €
Alimentation CC	Eléctronique	PS910	9V 1A	1	Cliquer ici	7,17 €	Gotronic	7,17 €
Level shifter	Eléctronique	CYT1076	Prix pour pack de 10 pièces – 1 suffit	1	Cliquer ici	5,99 €	Amazon	5,99 €
Powerbank	Eléctronique	U = 5V, I>=1A	à modifier pour souder deux fils en sortie (pas la place pour une prise)	1				5,00 €
Module régulateur de tension	Eléctronique	DRF0571	Tension entrée : 1.25V – 30V | Sortie : 5V	1	Cliquer ici	3,29 €	Gotronic	3,29 €
Connecteur GPIO femelle à souder	Consommable		40 pins (2*20)	1	Cliquer ici	2,58 €	Gotronic	2,58 €
Buzzer piezoelectrique	Eléctronique	THDZ		1	Cliquer ici	2,46 €	Gotronic	2,46 €

Encodeur rotatif	Eléctronique	KY40	Bouton intégré	1	Cliquer ici	1,83 €	Gotronic	1,83 €
Module RTC	Eléctronique	Module à base de DS1302	Prix pour pack de 5 pièces – 1 suffit	1	Cliquer ici	1,39 €	Amazon	1,39 €
Transistor NPN	Eléctronique	2N2222	hauteur >=6mm	1	Cliquer ici	0,25 €	Gotronic	0,25 €
Fixation LED	Mécanique		pour LED 5mm	1	Cliquer ici	0,17 €	Gotronic	0,17 €
Bouton 6x6mm	Eléctronique	KRS0610	hauteur 7mm	1	Cliquer ici	0,25 €	Gotronic	0,25 €
Led rouge	Eléctronique		3mm	1	Cliquer ici	0,13 €	Gotronic	0,13 €
IRF7319	Eléctronique	IRF7319	Composant à monter en surface	1	Cliquer ici	0,83 €	Digikey	0,83 €
MCP3001	Eléctronique	MCP3001	Convertisseur analogique numérique	1	Cliquer ici	1,51 €	Digikey	1,51 €
Entretoises taraudées	Mécanique		M3*6+6 – Nylon de préférence (pour fixations isolées)	8				0,00 €
Vis	Mécanique		M3*6	14				0,00 €
Résistances 1/4W	Consommable		1*680 R, 2*200 R,1*250 R, 1*500 R, 2*300K	5				0,00 €
Fil électrique souple, multibrin, diam 0,3	Consommable	30AWG	Rouge – Noir – Bleu – Vert – Jaune	5				0,00 €
PCB vierge	Consommable		Carré de 12mm de côté (pour la partie commande du ventilateur)	1				0,00 €

Tableau 1 : Liste du matériel électronique nécessaires pour la réalisation du boitier

L'élément le plus cher utilisé dans le projet est une carte Raspberry Pi 3 modèle B+. Il s'agit d'un nano-ordinateur très performant, de plus en plus utilisé en électronique pour ses ports GPIO[7] qui

[7] GPIO = General Purpose Input Output

permettent de communiquer avec une multitude de composants. Ce nano-ordinateur est particulièrement adapté à ce projet car ce dernier implique de la gestion de fichiers, une connexion à Internet filaire, et une communication avec clé USB. Or la Raspberry Pi 3 embarque déjà des ports USB, une prise Ethernet et, en tant que nano-ordinateur exécutant un système d'exploitation basé sur Linux, a déjà tout ce qu'il faut pour manipuler simplement des fichiers en lignes de commandes.

Nous aurions pu nous tourner vers une solution moins gourmande en énergie (type Arduino) mais il aurait alors fallu rajouter tous les modules nécessaires à nos communications. Finalement, le projet serait donc revenu bien plus cher et les performances auraient été très réduites (l'Arduino n'étant pas capable, par exemple, de gérer plusieurs processus en simultané, ce que la Raspberry sait faire).

En plus des composants électroniques, un accès à un FabLab ou un atelier est recommandé, dans la mesure où la réalisation du boîtier nécessite un accès aux machines suivantes :

- Imprimante 3D (min 100x100x150mm de volume imprimable)

Figure 3 : Imprimante 3D

- Fer à souder

Figure 4 : Station de soudure à l'étain

Schéma électronique

Voici le schéma électronique final du boîtier (celui-ci a été réalisé avec le logiciel Autodesk Eagle) :

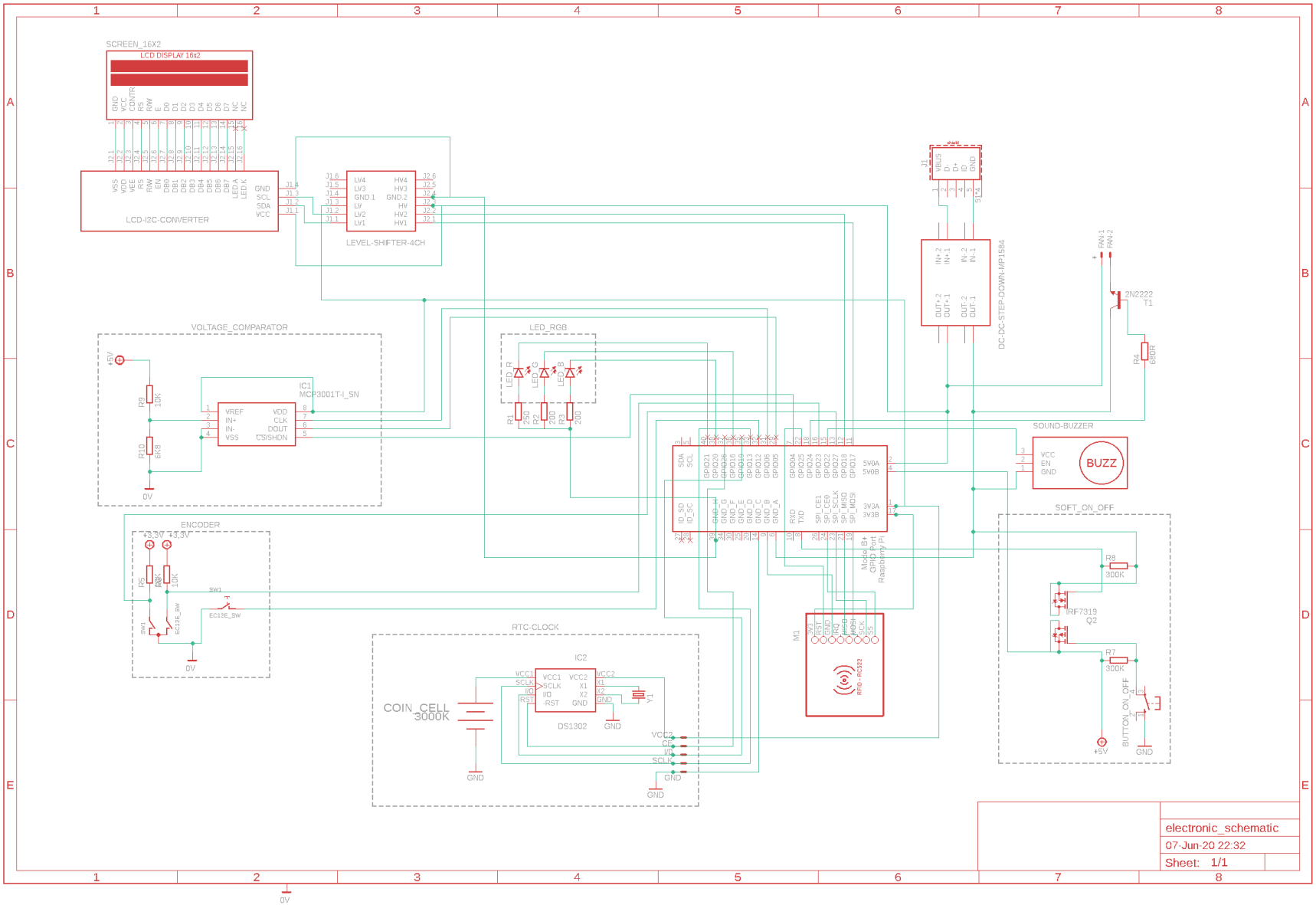


Figure 5 : Schéma électronique du boîtier

Cette version finale contient les ajouts liés à la détection du niveau de batterie et au circuit de gestion de l'alimentation qui n'étaient pas présents dans le rapport intermédiaire.

Modélisation et impression 3D du boîtier

Une étape importante du projet qui n'a pas été mentionnée dans le rapport intermédiaire a été la modélisation du boîtier. Celle-ci a été faite avec le logiciel Fusion 360. Voici le résultat de notre modélisation (téléchargeable sur notre dépôt Git ou sur GrabCad) :

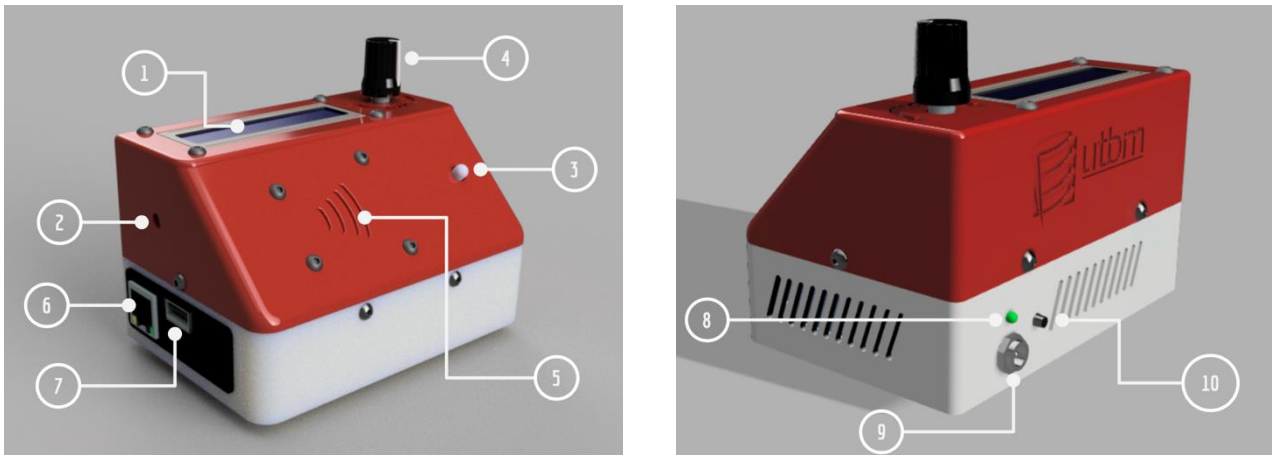


Figure 6 : Modélisation du boitier

1. Écran
2. Buzzer
3. LED d'état
4. Sélecteur de menu
5. Zone de scan
6. Port Ethernet
7. Prise USB
8. LED d'allumage
9. Prise d'alimentation
10. Bouton D

Le boitier a été imprimé en 3D sur nos machines personnelles.

Organisation de la programmation

L'ensemble du code source qui permet au boîtier de fonctionner a été écrit en langage Python, version 2.7. Il s'agit de la version préinstallée sur Raspbian (principal système d'exploitation utilisé pour les Raspberry Pi). Une partie non-négligeable des connaissances qui nous ont permis d'écrire ce code source (arbres, fonctions récursives) provient de l'UV LO21 de l'UTBM. Afin de se conformer aux bonnes pratiques liées à l'écriture de code, nous avons décidé d'écrire l'ensemble du code en anglais.

Notre dépôt Git

Conscients de la quantité de code à fournir (qui, bien qu'optimisée, s'élève finalement à plus de 2000 lignes) et afin d'appréhender la partie programmation informatique du projet dans les meilleures conditions, nous avons créé un dépôt Git sur GitHub. Git est un gestionnaire de versions de code. Un dépôt Git est un dossier contenant principalement du code, stocké sur un serveur, auquel chaque utilisateur authentifié (possédant un compte associé au projet) peut contribuer. Chaque développeur peut alors cloner ce git localement, y apporter les modifications qu'il souhaite, puis procéder à leur indexation (git commit) avant de les publier (git push).

Notre dépôt Git est hébergé à l'URL suivant : https://github.com/totordudu/UTBM_TZ20.

Nous avons également utilisé les fonctionnalités **Issues** et **Project** proposées par GitHub, pour notre organisation personnelle :

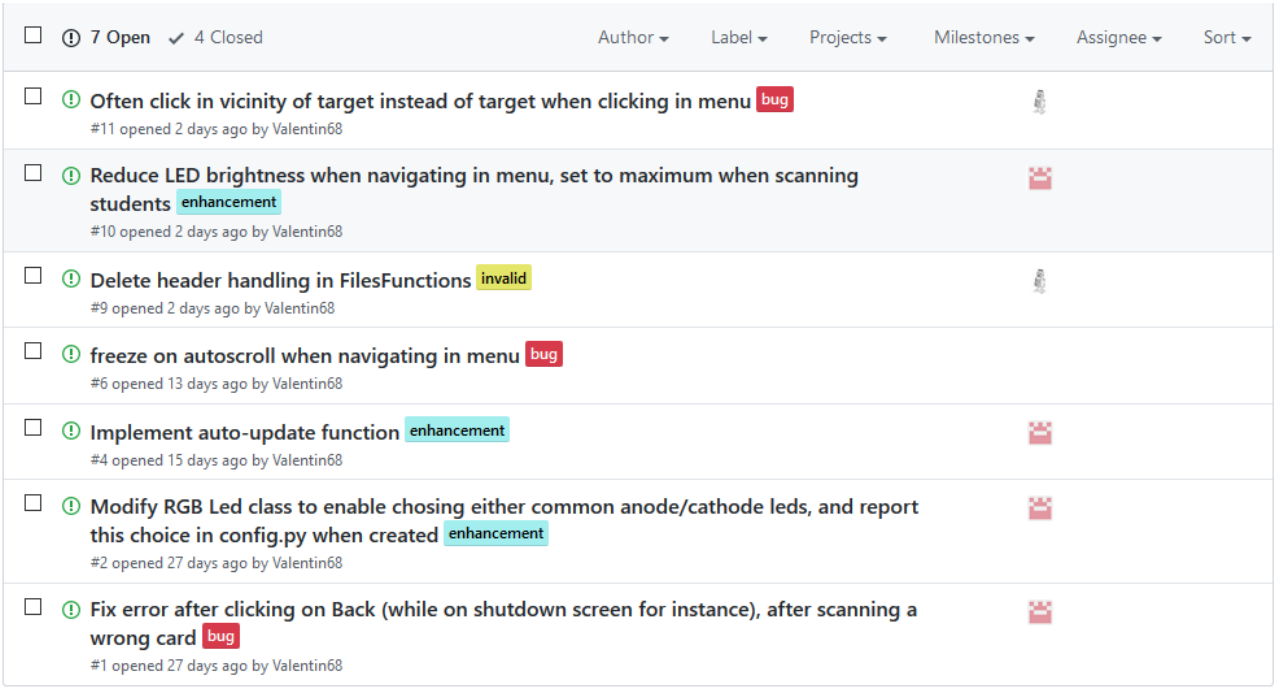


Figure 7 : Section "Issues" de notre dépôt Git (au 26/04/2020)

Depuis le rapport intermédiaire, l'ensemble des Issues importantes ont été traitées. D'autres, moins importantes ou qui auraient nécessité des modifications trop importantes au niveau du code ou de l'électroniques, ont été abandonnées.

Gestion des fichiers de listes

Pour une meilleure compréhension des différentes opérations de traitement des données (cartes, étudiants, contrôles) réalisées par le boitier, voici un algorigramme :

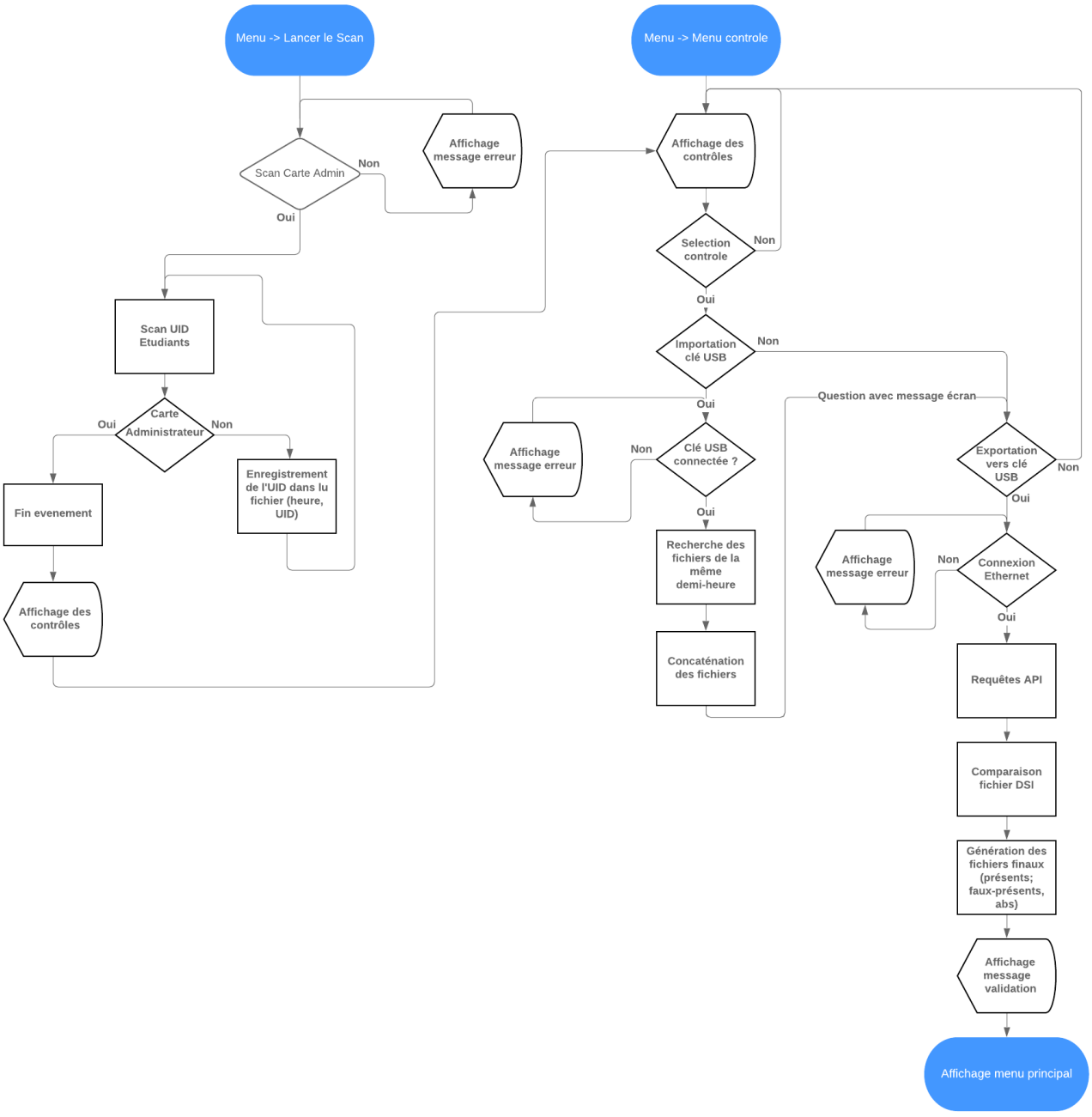


Figure 8 : Algorigramme de la gestion des fichiers telle qu'implémentée dans le boitier

Librairies utilisées

Plusieurs librairies Python ont été utilisées dans ce projet, toutes ont été trouvées sur GitHub :

- **lcd_lib** nous permet de gérer l'affichage d'informations sur l'écran LCD via l'interface I2C
- **pyRPiRTC** nous permet de gérer l'horloge externe (RTC[8]) en lecture et écriture
- **treelib** nous permet d'accéder à la structure informatique de l'arbre *n*-aire dans le cadre de notre menu (voir la partie Chemin utilisateur)

[8] RTC = Real Time Clock

- **GPIO** est une librairie spécifique à Raspbian, permettant de gérer les ports GPIO mentionnés précédemment (entrées/sorties)
- **MFRC522** nous permet de communiquer avec le module RC522 (module RFID permettant de scanner les cartes étudiantes)
- **datetime** nous permet d'accéder à la structure informatique éponyme, qui permet la gestion des ensembles date + heure
- **subprocess** nous permet de déclencher des sous-processus
- **os** nous permet de faire appel à des commandes système (normalement saisies dans le terminal)
- **signal** nous permet de capturer des signaux système (dans notre cas, SIGINT pour une interruption par Ctrl+C) pour les dérouter vers des fonctions appelées traitants (handler en anglais)
- **requests** nous permet d'envoyer des requêtes vers google pour tester la connexion du boîtier à internet
- **ConfigParser** nous permet de lire et de modifier le fichier de configuration contenant les paramètres du boîtier, pour les garder en mémoire permanente
- **sys** nous permet d'utiliser la commande exit() pour mettre fin à l'exécution du programme
- **pyudev** et **psutil** nous permettent de détecter le branchement d'une clé USB

Nous avons nous-mêmes codé plusieurs classes Python pour gérer certains éléments propres au boîtier :

- **buzzer** nous permet de jouer des sons prédéfinis (que nous avons nous-mêmes créés) selon la situation dans laquelle se trouve l'utilisateur
- **rgb_led** nous permet d'afficher les 7 couleurs avec la LED RGB[9], qui vient en complément de l'écran pour guider l'utilisateur dans le menu et les fonctions auxquelles ce dernier lui permet d'accéder
- **rotary_encoder** permet de gérer l'encodeur rotatif monté en façade supérieur du boîtier, seule entrée de l'IHM[10] mise en place sur le boîtier, à la fois en rotation et au clic.

Chemin utilisateur

Lors de la programmation d'un boîtier dont l'utilisation doit être simplifiée au maximum tout en assurant un maximum de tâches prédéfinies, il est important de bien réfléchir au chemin utilisateur : l'accessibilité aux différentes fonctions doit être épurée, claire et organisée. Pour ce faire, nous avons imaginé un menu sous forme d'arbre. L'arbre est une structure de données très employée en informatique dès qu'il est utile de hiérarchiser des données sous une arborescence. On parle d'arbre binaire (ou Btree) lorsque chaque nœud a 2 fils au plus, ou d'arbre n-aire dans le cas contraire.

[9] RGB = Red Green Blue, appliqué à une LED cela signifie qu'elle peut afficher n'importe quelle couleur par synthèse additive de la lumière

[10] IHM = Interface Homme/Machine

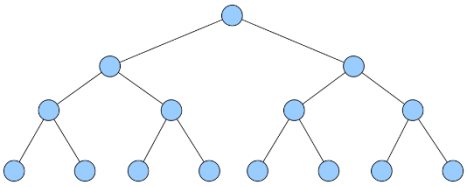


Figure 10 : Exemple d'arbre binaire

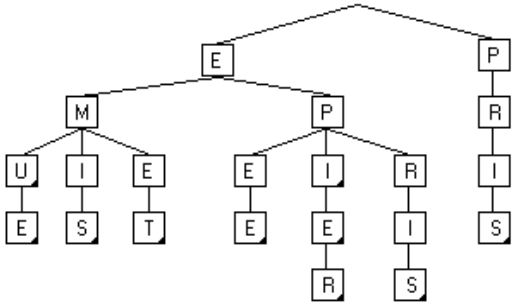


Figure 9 : Exemple d'arbre *n*-aire

Dans le cadre de notre menu, nous avons donc choisi d'employer la structure de l'arbre *n*-aire, prévoyant qu'une entrée de menu permettrait d'accéder à plus de 2 sous-menus. Nous avons essayé d'optimiser ce menu afin de mettre en avant les fonctions qui seront les plus utilisées. Voici l'arbre simplifié de notre menu (l'arbre complet, lui, contient les chaines de caractères à afficher sur l'écran pendant la navigation, la couleur à afficher sur la LED en façade -ici représentée par la couleur du fond du nœud- pour chaque sous-menu ainsi que le nom de la fonction terminale à appeler pour chaque feuille[11]) :

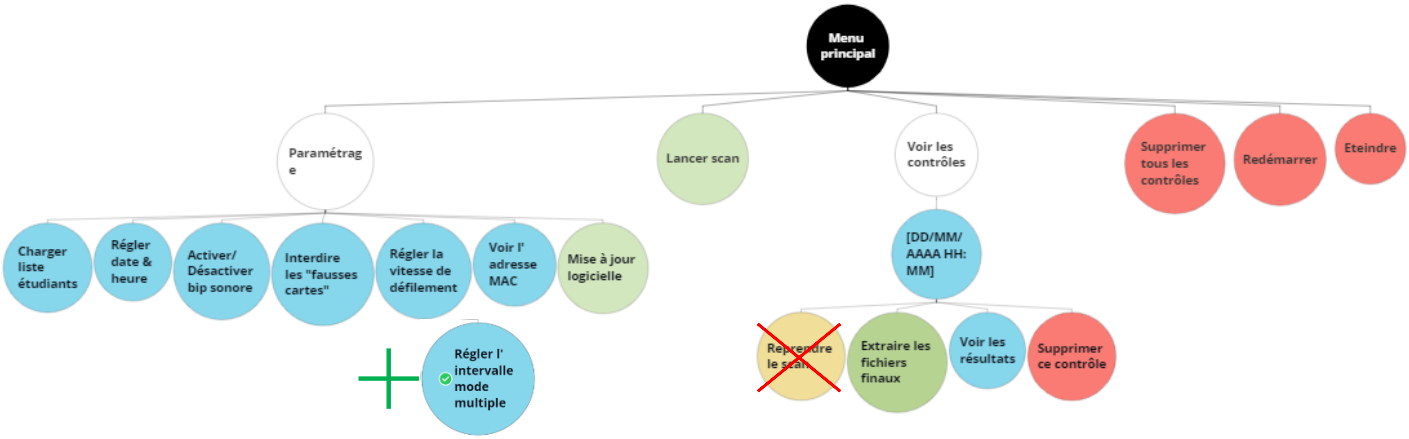


Figure 11 : Arbre simplifié du menu de navigation du boîtier

Depuis le rapport intermédiaire, nous avons supprimé la fonction « Reprendre le scan » accessible depuis le sous-menu « voir les contrôles » car elle est superflue : nous comptions l'implémenter pour permettre à l'administrateur de relancer un scan en cas d'interruption par manque de batterie par exemple. Toutefois, il lui suffirait dans ce cas de lancer le deuxième scan normalement, puis de procéder à une extraction en mode « boitiers multiples » avec fusion pour obtenir de résultat global.

Aussi, nous avons ajouté la fonction « Régler l'intervalle du mode multiple » qui permet d'ajuster l'intervalle de temps maximum dans lequel deux contrôles (l'un sur le boîtier, l'autre déjà extrait sur la clé) seront considérés comme similaires et sujets à une fusion des résultats. Pour mieux comprendre, lisez la section Extraction et visualisation des résultats.

Précisions supplémentaires

- Un paramètre « Interdire les fausses cartes » apparaît dans l'arbre du menu. Celui-ci permet d'activer ou de désactiver une restriction lors de la phase de scans : lorsqu'elle est

[11] Une feuille est un nœud de l'arbre qui n'a aucun fils (dans notre cas, cela correspond aux fonctions, que l'on désigne ici comme fonctions « terminales », auquel notre menu permet d'accéder)

activée, cette restriction empêche les étudiants de scanner d'autres objets compatibles RFID que leur carte étudiante. Toutefois, comme cette restriction se base sur le format des UID de cartes (format spécifique au fournisseur de l'UTBM), nous avons pensé qu'il pourrait être utile de permettre à l'utilisateur de lever cette restriction en cas de changement de fournisseur par exemple, ou bien s'il souhaite contrôler un jour la présence de personnes ne possédant pas de carte UTBM.

- Un paramètre « Voir l'adresse MAC » permet à l'utilisateur de connaître facilement l'adresse physique de la Raspberry. En effet, il est important de la connaître puisqu'elle doit être ajoutée par la DSI à la liste blanche des équipements autorisés à accéder au réseau filaire. Cette autorisation lui sera indispensable pour pouvoir utiliser l'API fournie par la DSI permettant de connaître les propriétaires des cartes étudiantes.
- Un paramètre « mise à jour logicielle » permet à l'utilisateur de mettre à jour notre programme automatiquement. Cela sera utile dans le futur si des bugs que nous n'aurions pas anticipés surviennent après un certain nombre d'utilisations
- Dans la section « Voir contrôles », les différents contrôles auxquels il est possible d'accéder sont identifiés par leur date de création au format DD/MM/AAAA HH:MM.
- En dernière position de chaque sous-menu, une option « RETOUR » permet de remonter au niveau supérieur.

Finalisation du projet

Cette partie présente les étapes de finalisation du projet qui ont été mises en œuvre depuis la rédaction du rapport intermédiaire.

Circuit de gestion de l'alimentation

Le principal élément manquant du schéma électronique publié dans notre rapport intermédiaire était la gestion de la batterie. En effet, nous avions soulevé le problème de la gestion de la batterie en faisant l'expérience d'alimenter la Raspberry Pi en la raccordant directement à la sortie de la batterie. Nous avions alors observé un gros défaut, qui constitue non seulement un trouble pouvant endommager la Raspberry sur le long terme, mais également une entrave à l'ergonomie d'utilisation attendue pour ce boitier : la Raspberry étant toujours alimentée (même si éteinte au niveau logiciel), celle-ci consomme et décharge la batterie jusqu'à son déchargement complet. Pendant ce déchargement, nous avions alors observé un fonctionnement indésirable : la Raspberry tente de s'allumer, puis est coupée en séquence de boot à cause de la chute de tension occasionnée par le démarrage, puis la tension revient à la normale, la Raspberry se rallume, et le cycle recommence jusqu'à épuisement de la batterie.

Nous rappelons par ailleurs qu'en aucun cas la Raspberry ne doit pouvoir être éteinte autrement que par notre programme : les étudiants ne doivent pas pouvoir éteindre le boitier pendant qu'ils scannent leurs cartes, seul l'administrateur peut le décider. C'est pourquoi nous avons intégré le bouton « D » (comme démarrage) dans notre boitier, voir figure 4.

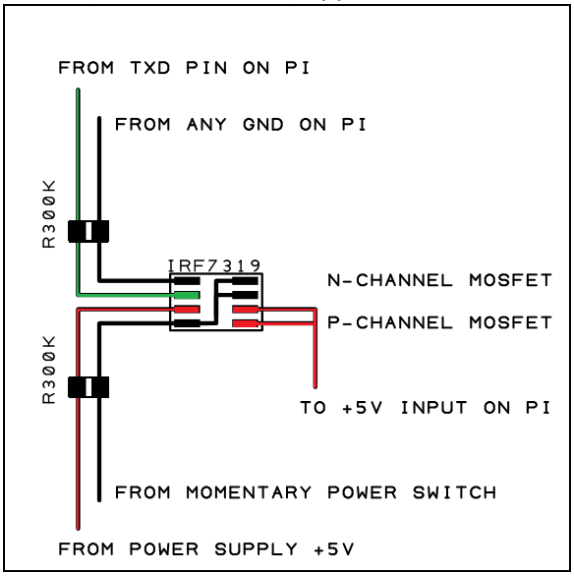
Nous avons alors établi une liste des fonctions que devrait remplir notre circuit :

- Détection du niveau de batterie et, si sa tension devient insuffisante pour alimenter la Raspberry de manière stable, on lui envoie un signal d'extinction (pour éviter une extinction à chaud, mauvaise pour la carte)
- Coupure de l'alimentation de la Raspberry lorsqu'elle passe en mode "shutdown" (dont l'origine peut être soit le point ci-dessus, soit une extinction volontaire) : il faut donc laisser à la Raspberry le temps de s'éteindre avant d'ouvrir le circuit qui l'alimente
- Réarmement de l'alimentation de la Raspberry par appui sur bouton poussoir. Ce bouton ne doit avoir d'effet que lorsque l'alimentation est coupée et doit alors la rétablir, mais il

ne doit pas la couper s'il est actionné et que la Raspberry est alimentée (fonctionnement en sens unique).

- Par ailleurs, le rechargement de la batterie (et donc dépassement du seuil de tension cité plus haut) ne doit pas rallumer la Raspberry, qui ne doit l'être que par un appui sur le bouton D
- Lorsque l'alimentation de la Raspberry est coupée, le circuit lui-même ne doit presque rien consommer (pour éviter d'avoir à recharger la batterie trop souvent)

Après plusieurs recherches infructueuses, nous avons finalement trouvé le circuit suivant (othermod, 2016) appelé Soft On/Off circuit :



Ce circuit fonctionne à partir d'un composant nommé IRF7319 qui n'est autre qu'un assemblage de deux transistors MOSFET, l'un à canal P et l'autre à canal N.
Le transistor à canal P est passant lorsque sa grille (gate) est reliée à la masse, donc lorsqu'on appuie sur le bouton D (Momentary Power Switch). Le second MOSFET, à canal N, permet de maintenir cet état passant : ce dernier n'est passant que lorsque sa grille est à l'état haut, ce qui est le cas lorsque la Raspberry est allumée puisque TXD vaut alors 1. La Raspberry est donc alimentée de manière stable.
La seule manière de couper cette alimentation est que TXD passe à l'état bas, ce qui arrive évidemment lorsque la Raspberry est en mode « shutdown » (éteinte logiciellement).

Ce circuit règle donc une bonne partie du problème, mais il reste la tension de la batterie à gérer : on doit être capables de la lire et de détecter qu'elle passe en dessous d'un certain seuil. La Raspberry Pi ne disposant pas de convertisseur Analogique/Numérique (ADC), nous avons utilisé l'ADC MCP3001 de la très réputée famille des MCP300x et l'avons associé à un pont diviseur dont nous avons judicieusement calculé les valeurs de résistances :
Puisqu'on s'attend à lire une tension maximum de 5v, et qu'on doit retrouver en sortie du pont diviseur une tension de 3v (on garde une marge de sécurité de 0.3v par rapport aux 3.3v de la Raspberry), cela nous donne l'équation suivante (du pont diviseur sans charge avec R1 valant arbitrairement 10K) :

$$S = \frac{R1}{R1+R2} * E \Leftrightarrow 3 = 5 * \frac{10000}{x+10000} \Rightarrow x \cong 6K8$$

Finalement, nous avons donc le circuit de contrôle de tension suivant :

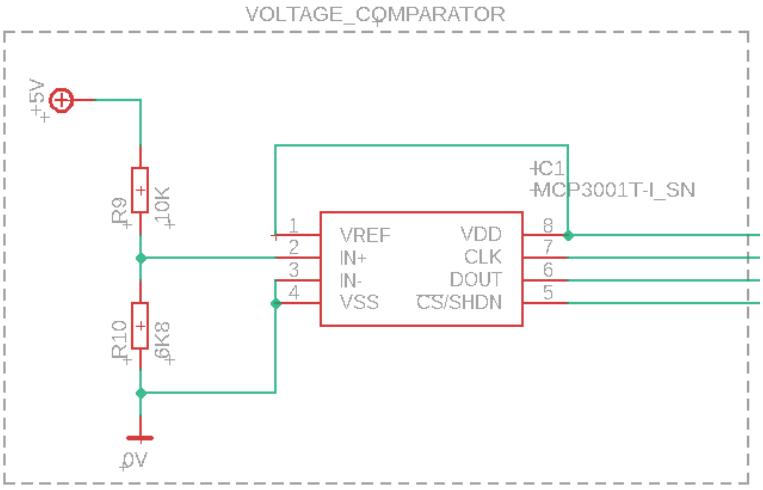


Figure 12 : Circuit de contrôle de la tension fournie par la batterie

Il nous suffit alors de créer un service sur la Raspberry dont la seule activité sera de lire régulièrement la tension fournie par la batterie, et de déclencher une extinction par *sudo shutdown now* si cette tension est inférieure à un seuil critique à déterminer.
Ce service (*battery.service*) et le code Python associé (*external_devices/read_voltage.py*) sont à retrouver dans le dépôt Git.

Extraction et visualisation des résultats

Concernant l'extraction des fichiers relatifs à un contrôle, nous avons suivi la figure 6 établie pour le rapport intermédiaire au détail près que nous laissons finalement le choix à l'administrateur de rentrer en mode « boitier unique » ou « boitier multiple » suivant qu'il veuille ou non réaliser une fusion par rapport à une extraction précédemment effectuée, dont les résultats sont sur une clé USB branchée à la Raspberry.

Conclusion

Pour conclure, nous sommes très satisfaits du boîtier que nous avons conçu. Nous pensons qu'il répond à tous les critères établis par le secrétariat du Tronc Communs, et attendons leurs premiers tests pour en avoir la confirmation. Mener un tel projet dans une période de confinement n'a pas été chose facile, mais en nous organisant correctement pour travailler efficacement à distance nous avons pu mener le projet à son aboutissement. Celui-ci nous a fait appel à nos connaissances à la fois en mécanique, en électronique et en informatique. Il nous a permis de s'échanger des connaissances et des compétences souvent complémentaires au sein du binôme.

Bibliographie

goobering. 2016. Measuring voltage and current of battery. *raspberrypi.stackexchange.* [En ligne] 6 Juillet 2016. [Citation : 3 Juin 2020.] https://raspberrypi.stackexchange.com/questions/50870/measuring-voltage-and-current-of-battery.

othermod. 2016. Raspberry Pi Simple Soft On/Off Circuit. *othermod.com.* [En ligne] 13 Juillet 2016. [Citation : 03 Juin 2020.] https://othermod.com/raspberry-pi-soft-onoff-circuit/.

wikipedia.fr. 2020. Radio-identification. *Wikipedia.* [En ligne] 7 avril 2020. [Citation : 20 avril 2020.] https://fr.wikipedia.org/wiki/Radio-identification.