

UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Rapport de projet

Rapport final

Conception d'un système embarqué et réseauté
GIF500

Présenté à
M. Domingo Palao Muñoz

Présenté par l'équipe 2 :

Mathieu DOSTIE	DOSM2902
Émile FUGULIN	FUGE2701
Philippe GIRARD	GIRP2705
Damien HULMANN	HULD1501
Julien LAROCHELLE	LARJ2526
Samir LECHEKHAB	LECS2813
Donavan MARTIN	MARD1206

Table des matières

1	Présentation du prototype	1
2	Cahier des charges	2
3	Branchement des différents éléments	6
3.1	Général	6
3.2	Pont	6
3.3	Terminal de paiement	7
3.4	Terminal de recharge	7
4	Procédure de test du système du prototype	10
4.1	Code C	10
4.2	Code Javascript	10
4.3	Assurance qualité	10
5	Explications techniques des fonctions et composants	14
5.1	Pont	14
5.1.1	Fonctionnement	14
5.2	Terminal de recharge	14
5.2.1	Fonctionnement	14
5.2.2	Protocole	15
5.3	Terminal de paiement	17
5.3.1	Fonctionnement	17
5.3.2	Protocole	19
5.4	Serveur et base de données	19
5.4.1	API	19
5.4.2	Site Web	20
5.4.3	Postgres avec tables	20
6	Structure du code du projet	23
6.1	Code embarqué	23
6.2	Code serveur	23
7	Réflexion sur les corrections à apporter à la 2^e itération	24
8	Gestion de projet	25
8.1	Courbe en S	25
8.2	Explication de l'allure des courbes	25
9	Gestion des risques	26
10	Améliorations futures	29

1 Présentation du prototype

La première itération du prototype suit les requis établis en début de session. En effet, il comprend trois grandes parties : le terminal de recharge, le terminal de paiement et le pont/serveur.

Premièrement, à partir du terminal de recharge l'utilisateur peut savoir l'état de son compte et faire un dépôt d'argent. Le prototype accepte seulement les pièces de monnaie. Pour être en mesure d'effectuer une recharge, l'utilisateur doit présenter sa carte NFC. Le prototype est transportable et fonctionnel sur batterie, ce qui est utile pour l'avènement d'événements spontanés, comme les 5@8.

Deuxièmement, le terminal de paiement est le lien entre les consommateurs et les marchands. En effet, le marchand entre les articles achetés par le client pour obtenir le prix total des achats. Ensuite, le client doit présenter sa carte NFC pour effectuer le paiement. Une DEL et un écran transmettent les informations reliées aux transactions. Ce terminal fonctionne aussi sur batterie et il est facile d'installation, étant donnée sa forme compacte.

Troisièmement, le pont permet de transmettre les requêtes des terminaux au serveur pour effectuer les bonnes requêtes et obtenir les informations nécessaires. Un site internet permet aux utilisateurs de voir leur état de compte en temps réel. Pour les marchands, ils ont la possibilité de modifier leur liste de produits offerts et de voir leurs ventes. Pour les clients, ils ont la possibilité de voir leurs achats et leur solde de compte. Ce site web est disponible à tout moment.

2 Cahier des charges

Le cahier des charges contient une échelle de priorité dont voici les différents niveaux :

1. Fonctionnalités requises à la réussite du projet ;
2. Fonctionnalités primaires ;
3. Fonctionnalités secondaires ;
4. Fonctionnalités facultatives (celles qui ne seront probablement pas implémentées).

TABLEAU 2-1 – Fonctionnalités générales

Code	Fonctionnalité	Objectif	Description	Contraintes	Priorité	Implanté
1.1	Avoir des nœuds mobiles	Avoir des stations mobiles	Les stations de paiement doivent être autonomes	Utilisation d'une batterie et communication Zigbee	1	✓
1.2	Avoir un nœud fixe	Avoir un pont entre le serveur et les nœuds mobiles	La station fixe doit acheminer les messages entre les nœuds mobiles et le serveur	Communique en TCP/IP et Zigbee	1	✓
1.3	Communication Zigbee	Les nœuds mobiles et fixes doivent communiquer de façon sans-fil	Doit fonctionner dans un rayon moins de 40 mètres	La distance séparant 2 nœuds	1	✓

TABLEAU 2-2 – Fonctionnalités du terminal de paiement

Code	Fonctionnalité	Objectif	Description	Contraintes	Priorité	Implanté
2.1	Effectuer un paiement	Payer un article	Utiliser l'argent du compte client pour payer un produit	1 paiement par 2-3 secondes. Argent canadien.	1	✓
2.2	Fonctionner sur batterie	Le terminal de paiement doit être portable	Le terminal peut être facilement déplacé	Durée : environ 12h	1	✓
2.3	Entrer le numéron de l'article acheté et la quantité	Définir un prix pour chaque article dans la BD	L'utilisateur doit être en mesure d'écrire le la quantité achetée de chaque article	Entre 1 et 9 produits par marchand	2	✓
2.4	Afficher l'état du paiement	Connaître l'état du paiement	L'utilisateur doit être assuré que le paiement à fonctionner	Au moins les informations sur autoriser ou non	2	✓
2.5	Lire le numéro d'un puce NFC	Identifier le client	Associer le numéro de carte à un compte dans la base de données	Doit être capable de lire la carte à 1 cm	3	✓
2.6	Afficher le prix total de la transaction	Connaître le prix total du panier	L'utilisateur doit être en mesure de savoir le prix payé pour son achat	Montrer jusqu'au 5 sous. Max 999\$	3	✓
2.7	Détecter la position du terminal	Connaître l'emplacement du terminal	Être en mesure de connaître l'emplacement du terminal à tout moment	6-7 mètres près	4	

TABLEAU 2-3 – Fonctionnalités du poste de recharge

Code	Fonctionnalité	Objectif	Description	Contraintes	Priorité	Implanté
3.1	Effectuer une recharge	Déposé de l'argent dans le compte d'un usager	Les usagers doivent être en mesure	Doit être capable de prendre les pièces (2, 1, 25€, 10€, 5€)	1	✓
3.2	Afficher l'état du compte	Connaître le balance du compte d'un usager	L'utilisateur doit être capable de savoir comment il a d'argent dans son compte	Jusqu'au 5 €	2	✓
3.3	Compteur d'argent	Être capable de connaître la valeur de l'argent déposé	Doit compter la valeur de l'argent déposé par l'utilisateur	Compter les pièces	2	✓
3.4	Fonctionner sur batterie	Le terminal de paiement doit être portable	Le terminal peut être facilement déplacé	Durée : environ 4 mois	2	✓
3.5	Lire le numéro d'une puce NFC	Identifier le client	Associer le numéro de carte à un compte dans la base de données	Doit être capable de lire la carte à 1 cm	3	✓

TABLEAU 2-4 – Fonctionnalités du serveur

Code	Fonctionnalité	Objectif	Description	Contraintes	Priorité	Implanté
4.1	Autorisation des paiements	Associer une transaction au bon compte	Doit être capable de savoir si le compte a assez d'argent	État autorisé ou non	1	✓
4.2	Autorisation des recharges	Associer un dépôt d'argent au bon compte	Doit être capable d'ajouter le montant déposé au bon compte	État autorisé ou non	1	✓
4.3	Persistance des transactions et des états de compte	Être en mesure d'avoir un état du compte à tout moment	Le système doit enregistrer toutes les transactions reliées à un compte	À vie – Procédure de backup possible	1	✓
4.4	Interface Web pour les usagers	Faire le suivi de ses achats	L'utilisateur doit être en mesure de voir toutes les achats qu'il a fait	Affichage du prix, de la quantité et de l'endroit	2	✓
4.5	Interface Web pour les vendeurs	Faire le suivi de ses ventes	Le vendeur doit être en mesure de voir toutes les ventes qu'il a fait	Affichage du prix, de la quantité et de l'endroit (ou client)	3	✓
4.6	Protéger le réseau contre la falsification	Sécuriser les transactions entre les différents noeuds	Utiliser une connexion sécurisée pour éviter que quelqu'un vole des données ou de l'argent	Encryption des communications	4	

3 Branchement des différents éléments

3.1 Général

Afin de faire fonctionner le système, tous les éléments sur le schéma 3-1 doivent être branchés. Il faut suivre l'ordre suivant afin que tout fonctionne correctement :

1. Alimenter le routeur fourni et attendre qu'il soit démarré
2. Brancher deux câbles ethernet sur des ports libres (gris) du routeur
3. Brancher un câble ethernet dans le raspberry pi et l'autre dans le pont
4. Alimenter le raspberry pi et le pont, attendre un moment qu'ils démarrent
5. Alimenter les deux terminaux, ils se connecteront au réseau automatiquement

Note : La base de données sur le raspberry pi démarre automatiquement, mais pas l'application « serveur », il faudra suivre la méthode suivante :

1. Connecter en SSH avec `pi:raspberrypi`
2. Naviguer vers `/ByPass/server`
3. Exécuter la commande `pm2 start server.js`

3.2 Pont

Le branchement du pont est relativement simple. Le LPC est connecté au port RJ45 afin de communiquer avec le serveur. Il est aussi connecté au module Xbee Contrôleur afin de recevoir les requêtes des terminaux et transmettre les réponses du serveur.

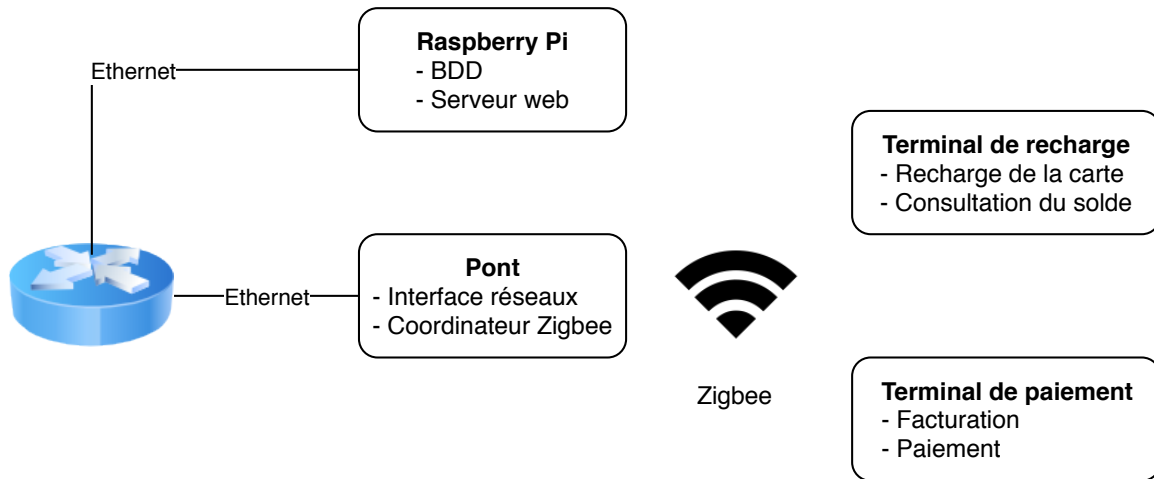


FIGURE 3-1 – Schéma-bloc des composantes générales du prototype

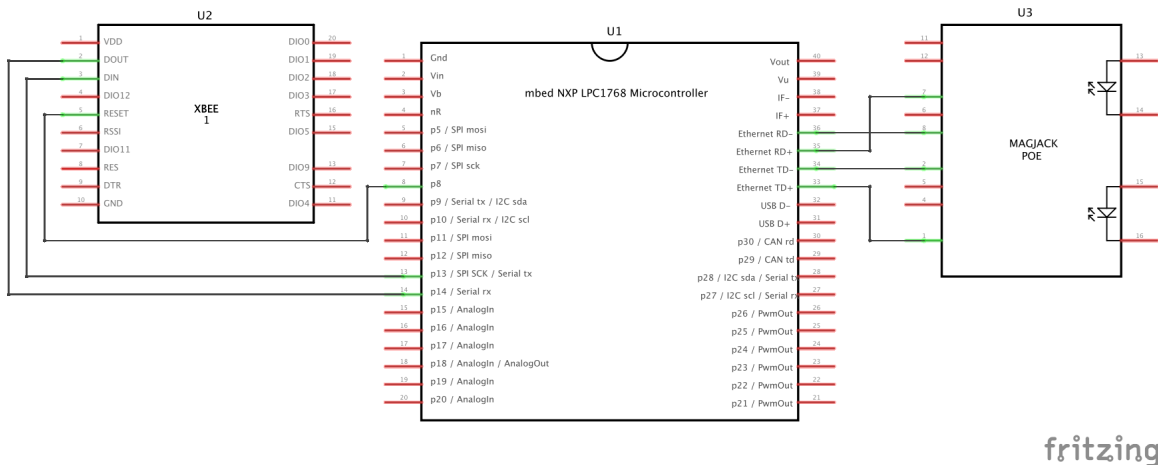


FIGURE 3-2 – Schéma des branchements du pont

3.3 Terminal de paiement

L'alimentation se fait à grâce à une batterie Li-ion sa tension nominal étant beaucoup trop élevée par rapport à notre circuit nous utilisons un régulateur de tension à découpage de type buck (MP1564) pour abaisser la tension à 5VDC, ce type de régulateur a un excellent rendement ce qui est une caractéristique recherchée dans un application embarquée, le XBee et le détecteur RFID MFRC522 n'acceptent cependant pas le 5VDC, nous utilisons donc le régulateur de tensions 3.3V du LCP1768 pour son alimentation.

Le LCD est alimenté en 5VDC et est commandé par un interface de 4 bits en parallèles pour les données et de 2 pins pour l'activation du module et une pour lui indiquer s'il s'agit d'une commande ou d'un caractère.

Le détecteur RFID est interfacé en SPI (mosi, miso, sck et le ss) de plus une pin (IRQ) nous indique quand une carte est détectée et une pin pour le reset. Une fois la carte détectée le module nous retourne son identifiant via SPI.

Finalement le XBee est interfacé en UART et possède une pin de plus pour le reset.

3.4 Terminal de recharge

Tout comme le terminal de paiement l'alimentation se fait grâce à une batterie Li-ion. Sur ce montage, nous avons ajouté un régulateur de tension 12 V pour l'alimentation du monnayeur, étant donné que nous n'avions à notre disposition qu'un régulateur de tension linéaire de type LM317. L'inconvénient avec ce type de régulateur est que beaucoup de puissance est dissipée dans ce dernier ; c'est pour cela que nous avons rajouté le transistor MOSFET 2N7000 pour alimenter ou non le monnayeur. En appliquant une tension sur la *gate* du 2N7000, nous connectons l'alimentation négative du monnayeur au GND, et ceci nous permet d'économiser beaucoup d'énergie. Le monnayeur quant à lui est un module qui se configure à l'aide d'une interface utilisateur sommaire, une fois configuré et lorsqu'une pièce est détectée, le monnayeur nous délivre N flancs descendants (N correspondant à la valeur de la pièce que nous avons configurée).

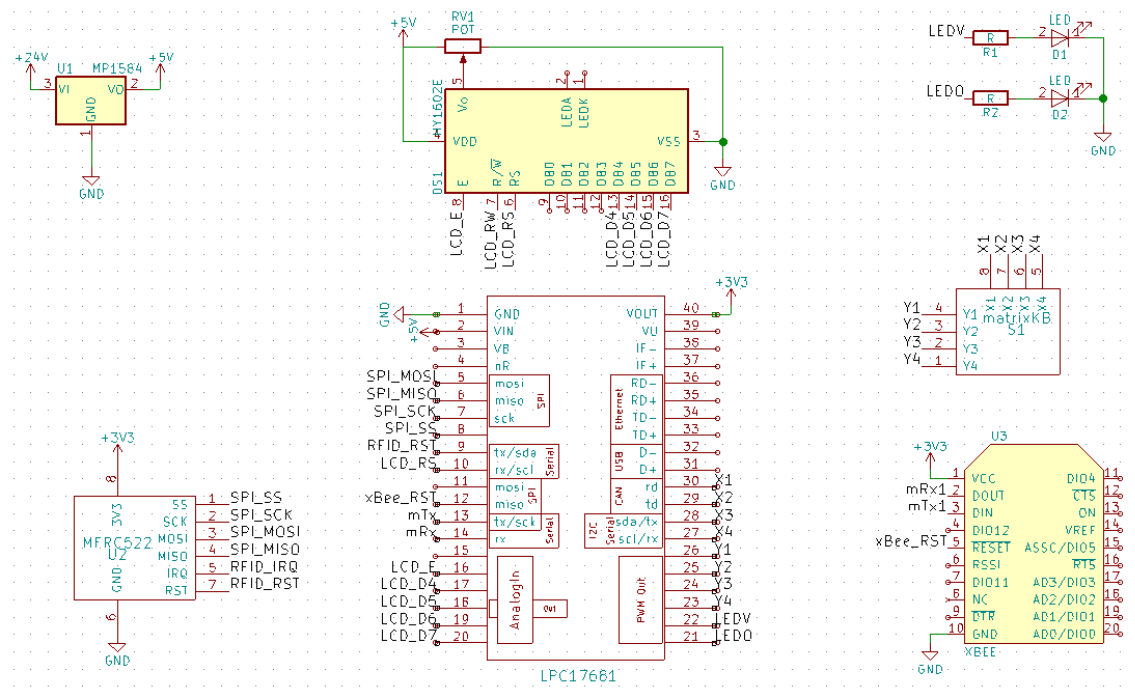


FIGURE 3-3 – Schéma des branchements du terminal de paiement

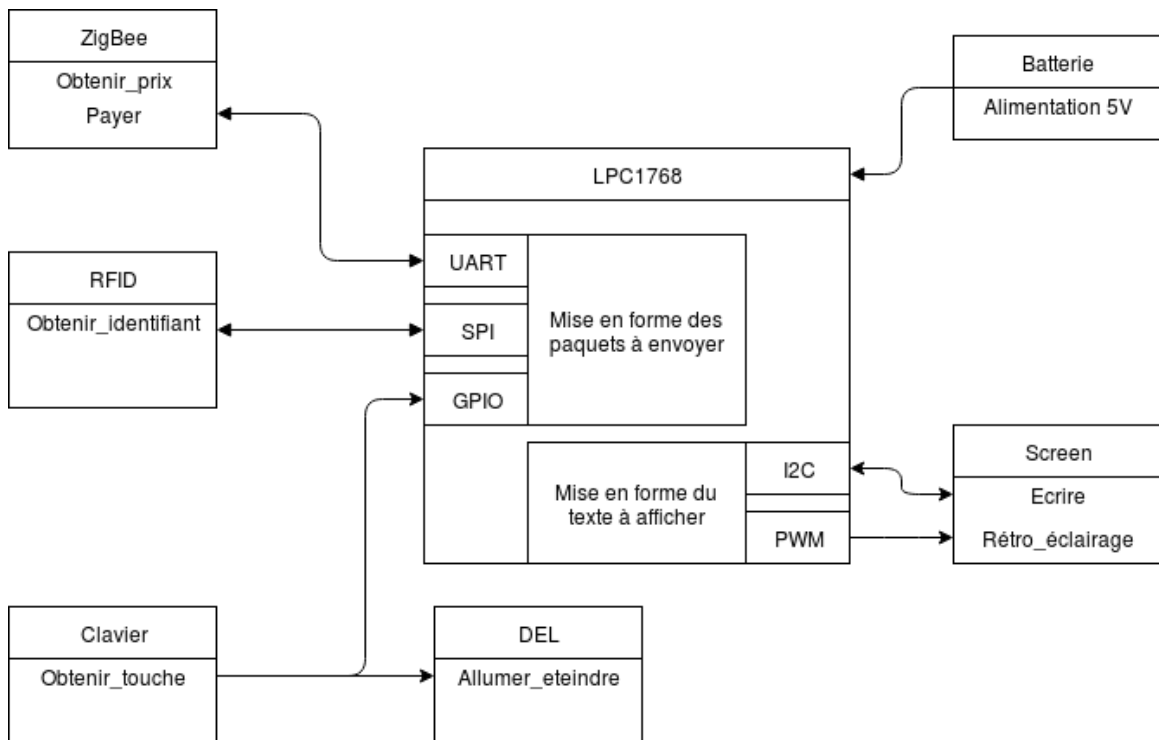


FIGURE 3-4 – Schéma-bloc du terminal de paiement

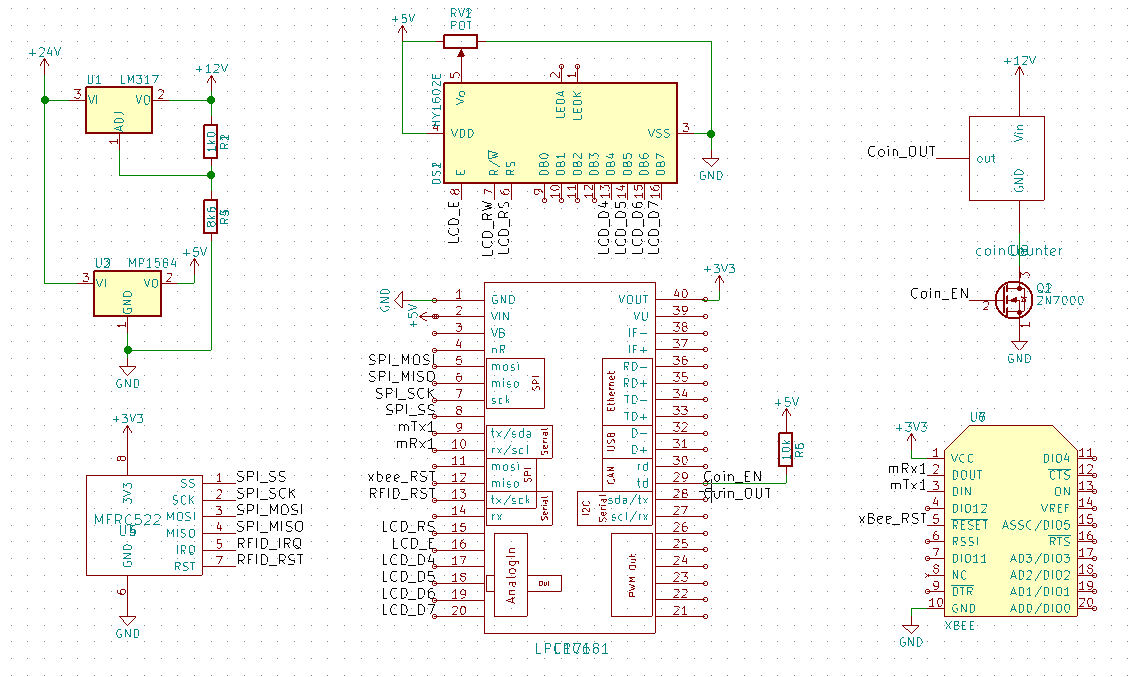


FIGURE 3-5 – Schéma des branchements du terminal de recharge

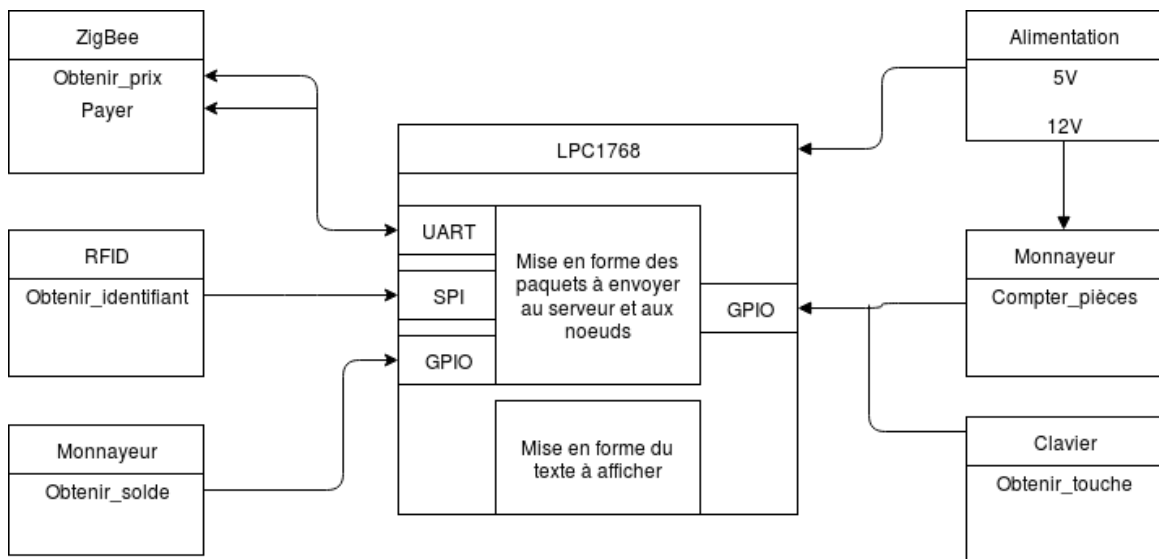


FIGURE 3-6 – Schéma-bloc du terminal de recharge

4 Procédure de test du système du prototype

4.1 Code C

Conséquemment aux ressources limitées du microcontrôleur, les tests unitaires et fonctionnels sont effectués lors d'une recompilation avec un « flag » de test d'activité. Lorsque ce « flag » est activé, une macro active des parties de code qui impriment des informations de déverminage à l'écran.

Il est facile avec les informations de déverminage de détecter les valeurs erronées. Cette technique permet de cerner rapidement les sections de code problématiques lorsqu'un bug est détecté.

4.2 Code Javascript

Du côté du serveur javascript un « framework » de test automatisé a été utilisé. L'équipe a utilisé une technique de développement TDD conforme aux règles de l'art de l'industrie. Le développement se faisait de la façon suivante :

1. Écriture d'un test d'intégration d'une fonctionnalité
2. Écriture de la fonctionnalité
3. Amélioration cyclique en continu

L'avantage de ce type de développement est qu'il permet à chaque développeur d'être agnostique du travail des autres. En effet, chaque développeur peut exécuter la commande « npm test » pour que la suite de tests s'effectue. Si tous les tests passent, cela indique qu'ils n'ont pas brisé une fonctionnalité et qu'ils peuvent envoyer leur code sur github.

La configuration du « framework » de tests et des environnements de tests a été très long. Cependant, maintenant que c'est configuré, la productivité a nettement augmenté. L'équipe a donc été gagnante d'utiliser le TDD pour développer le projet.

4.3 Assurance qualité

TABLEAU 4-5 – Plan de test du prototype

Test	Procédure	Résultat attendu
Initialisation	Envoie de commande et de caractère avant l'initialisation	Rien ne s'affiche
Affichage d'une chaîne de caractère	"Envoie de la trame "Borne de Paiement" grâce à la fonction d'envoi"	La trame est bien affichée
Mise en veille	Mettre la pin correspondante à la DEL du rétro-éclairage à l'état bas	Affichage est en veille et consomme moins de courant

Sortie 5V	Alimenter l'alimentation de 9V à 15V et contrôle de l'oscillation résiduelle à l'oscilloscope à vide et avec une charge de 500mA (résistive)	La sortie est stable à 5V
Prise en charge du mode veille	Utiliser la fonction de mise en veille implémentée par MBED	Le MBED est en mode veille
Acquisition d'un caractère	Appuyer sur une touche et décoder le caractère reçu par les GPIO	Le caractère est le bon
Tester la communication client vers pont	Envoyer une trame simple du zigbee client (type routeur) vers le pont (type coordinateur)	Réception de la trame sur le coordinateur
Tester la communication pont vers client	Envoyer une trame de réponse du pont vers client lors d'une requête	Réception de la trame sur le client
Reconnaissance des pièces	Insertion des pièces devant être reconnues et décodage du signal reçu	Le montant reçu est celui inséré
Acquisition de la position	Décoder la trame reçue par la communication série	Latitude et longitude
Mise en veille	Mettre la pin de commande du transistor en haute impédance	GPS est en veille et consomme moins de courant
Reconnaissance de badge	Passer un badge à 1cm du lecteur	ID de carte lues par le SPI
Mise en veille	Mettre la pin NRSTPD à '0'	NFC en veille et ne consomme pas de courant

Security des dépendances	"À partir d'un terminal aller au dossier "server/" et effectuer les commandes suivantes : nsp check ; snyk test ; snyk wizard ;"	Aucune faille de sécurité détectée
Création d'un compte	"POST /api/accounts "name" : "toto", "type" : "MERCHANT""	Le serveur retourne un JSON contenant les données du nouveau utilisateur
Login	"POST /api/accounts/login "name" : "toto", "type" : "MERCHANT""	Le serveur retourne le ID du compte
Information du compte	GET /api/accounts/:id	Le serveur retourne les informations du compte
Informations des comptes	GET /api/accounts	Le serveur retourne la liste de tous les utilisateurs.
Ajout item	"POST /api/items "merchant" : "...", "name" : "product1", "shortcut" : 1, "description" : "Toto", "price" : "100000""	Le serveur retourne les informations du nouveau produit
Ajout transaction	"POST /api/transactions "client" : "...", "merchant" : "...", "lineItems" : [{"itemId" : "...", "quantity" : 1.0}]"	Le serveur retourne les informations de la transaction
Solde par Zigbee	"POST /api/zigbee/bridge "Id" : NBR, "method" : 2, "clientId" : 16 char"	Le serveur retourne le solde du compte
Ajout argent par Zigbee	"POST /api/zigbee/bridge "Id" : NDR, "method" : 3, "clientId" : 16 char, "Amount" : NBR"	Le serveur retourne le nouveau solde du compte

Total commande par Zigbee	"POST /api/zigbee/bridge "id" : NBR, "method" : 4, "merchantId" : NBR, "items" : [11], "qty" : [1]"	Le serveur retourne le total de la commande
Tester la communication zigbee vers serveur	Envoyer une requête HTTP Post du pont vers le serveur	Réception de la trame sur le serveur
Tester la communication client vers serveur	Envoi d'une trame par zigbee du client	Réception de la trame sur le serveur
Tester la communication serveur vers client	Réponse du serveur lors d'un réception de trame du client	Réception de la trame sur le client
Transmettre les erreurs du serveur vers les noeuds	Envoyer une trame erronée d'un noeud	Réception d'une trame d'erreur sur le noeud
Roulement de la MEF	Faire rouler le squelette du code du terminal de paiement	Les bons états s'exécutent (confirmé par des printf)
Ajout d'un article dans le panier	Le marchand entre 1 article dans le panier	Le terminal affiche le prix de l'article vendu sur l'écran
Ajout de plus d'un article dans le panier (3)	Le marchand entre 3 articles différents dans le panier	Le terminal affiche le prix total des articles à l'écran
Paieement d'un panier (avec assez d'argent dans le compte)	L'utilisateur présente sa carte RFID	Le terminal effectue le paiement et affiche l'état du paiement à l'aide de DEL et à l'écran
Paieement d'un panier (avec solde insuffisant)	L'utilisateur présente sa carte RFID	Le terminal n'effectue pas le paiement et affiche l'état du paiement à l'aide de DEL et à l'écran
Roulement de la MEF	Faire rouler le squelette du code du terminal de recharge	Les bons états s'exécutent (confirmé par des printf)
Lecture de la carte RFID	Présenter une carte valide devant le lecteur RFID	Le lecteur détecte la carte et il associe le bon compte à la carte

5 Explications techniques des fonctions et composants

5.1 Pont

La figure 5-1 représente le diagramme de séquence du pont.

5.1.1 Fonctionnement

Le pont est composé de trois *threads*. Le `main` démarre le module d'éthernet et attend qu'il soit connecté au routeur. Il démarre ensuite les deux *threads* de communication. Finalement, il se met en veille et sert seulement au *debug* par la suite.

Le premier *thread* de communication est celui du Xbee. La librairie officielle de Digi pour Xbee est utilisée dans ce *thread*. Le *thread* traite les trames d'entrées et les places dans la *mailbox* du *thread* de communication Ethernet. Il écoute également de façon non bloquante sur sa propre *mailbox* et envoie les messages aux Xbee correspondants lorsque nécessaire.

Le deuxième *thread* de communication est celui du Ethernet. Dans celui-ci, une librairie tierce HTTP est utilisée avec quelques modifications internes afin de transmettre et recevoir du JSON. Lorsque ce *thread* reçoit un message du *thread* de communication Xbee, il effectue une requête POST vers le serveur et attend une réponse (avec un *timeout* de 1.5 secondes). Lorsque la requête revient, la réponse est transmise au *thread* de communication Xbee. En cas d'erreur, un message d'erreur personnalisé est envoyé au *thread* de communication Xbee. Tout est synchrone dans le *thread* de communication Ethernet.

Il est important de noter que le programme ne (dé)séréalise pas le JSON qu'il reçoit du serveur ou des Xbee. Cela est dû au fait qu'il manque de mémoire sur le MBED à cause de la bibliothèque Ethernet. Ainsi toutes les requêtes HTTP sont transmises au serveur sur le même « endpoint » et les réponses sont transmises intégralement aux terminaux qui eux, s'occuperont de les traiter.

5.2 Terminal de recharge

La figure 5-2 représente le diagramme de séquence du terminal de recharge.

5.2.1 Fonctionnement

Le terminal de recharge possède une architecture et un fonctionnement simple, basé sur une approche séquentielle. En effet, comme la réussite d'une recharge nécessite une série d'événements précis, le terminal est basé sur une machine à états finis (MEF) qui comporte 4 états.

Lorsque le terminal est mis sous tension, il s'ouvre dans l'état par défaut, soit l'état RFID. Le programme attend dans une fonction bloquante qu'un usager tape sa carte sur le lecteur RFID présent sur le devant du montage. Le numéro de la carte lue est gardé en mémoire.

Une fois que le lecteur détecte une carte, le programme change l'état pour `GET_CLIENT_INFO`. Dans cet état, le programme utilise une librairie JSON pour construire un message recevable par le serveur et il l'envoie au pont à l'aide du module XBEE. Si tout s'est déroulé correctement, le programme reçoit une réponse du serveur avec le solde courant du client. Le solde est affiché sur l'écran présent sur le terminal. Le programme change alors l'état pour `MONNAYEUR`.

Dans cet état, le programme alimente le monnayeur en courant, ce qui lui permet de recevoir des pièces de monnaie. Il est capable de prendre en entrée uniquement les pièces de 5, 10 et 25 cents, en plus des 1 et 2 dollars. Le programme reste actif à recevoir les pièces tant et aussi longtemps que l'utilisateur n'appuie pas sur le bouton présent sur le devant du montage. Le monnayeur envoie une série d'impulsions différentes selon la pièce déposée. C'est en comptant les fronts descendants que le programme est capable de savoir quelle pièce a été déposée. Une fois que l'utilisateur appuie sur le bouton, le programme change l'état pour `SEND_DATA` et éteint l'alimentation du monnayeur, pour économiser de l'énergie.

Le dernier état de la MEF sert à ajouter le montant déposé au solde de l'utilisateur. Pour ce faire, le programme utilise encore la librairie JSON pour construire un message qu'il envoie via le module XBEE au serveur. Si l'envoi se passe bien, le programme reçoit une réponse du serveur avec le nouveau solde de l'utilisateur. Le programme affiche le nouveau solde à l'écran et retombe dans son état initial, en attente d'un nouvel utilisateur.

5.2.2 Protocole

Le terminal de recharge utilise deux types de messages JSON sur les quatre disponibles. Dans tous les cas, les messages des terminaux possèdent les champs « id » (pour associer les réponses aux requêtes) et « method » (afin d'indiquer au serveur la requête à effectuer). Il est également important de noter que tous les nombres sont des entiers et qu'il faut donc diviser par 100 pour tout ce qui est relié à l'argent.

Ajout d'argent Cette requête permet au terminal de recharge d'ajouter de l'argent au compte de l'utilisateur.

Requête { " id ": NOMBRE, " method ": 3, " clientId ": STRING, " amount ": NOMBRE }	Réponse { " id ": NOMBRE, " balance ": NOMBRE }
---	---

Solde Cette requête permet au programme de récupérer le solde de l'utilisateur.

Requête { " id ": NOMBRE, " method ": 2, " clientId ": STRING }	Réponse { " id ": NOMBRE, " balance ": NOMBRE }
--	---

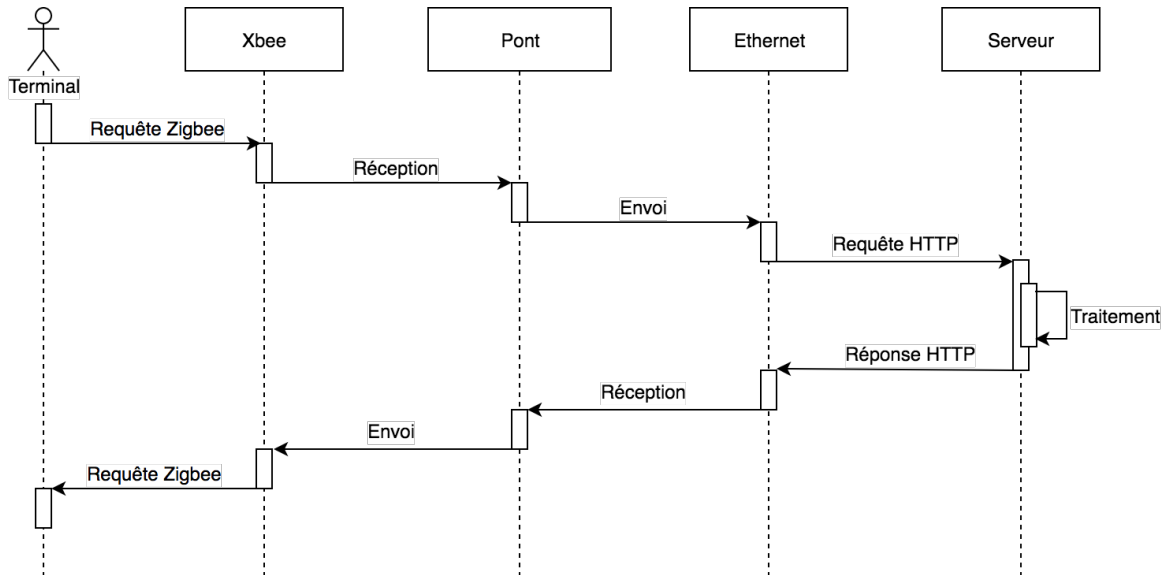


FIGURE 5-1 – Diagramme de séquence du pont

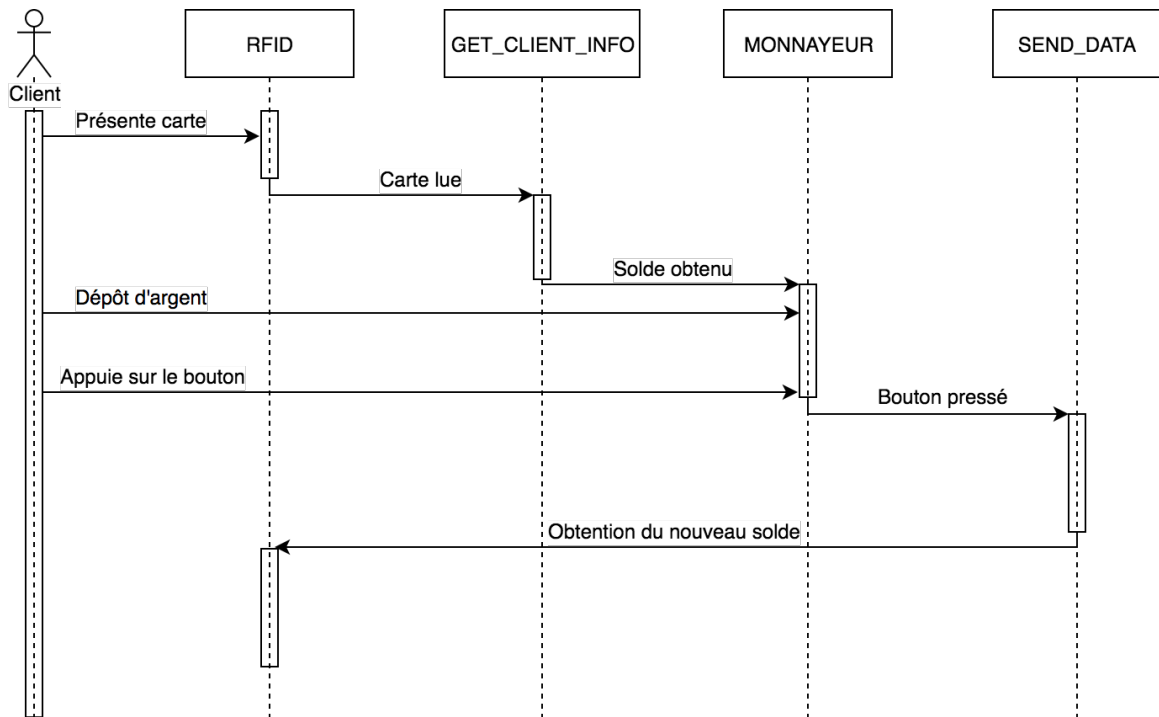


FIGURE 5-2 – Diagramme de séquence du terminal de recharge

5.3 Terminal de paiement

La figure 5-3 représente le diagramme de séquence du terminal de paiement.

5.3.1 Fonctionnement

Le terminal de paiement partage quelques similarités avec le terminal de recharge. Son fonctionnement est très séquentiel pour la présente itération. Il est aussi basé sur une machine à états finis (MEF) à 5 états représentés dans une « case/switch ».

Le premier état au démarrage est `GET_MERCHANT_ID`. Dans cet état, le programme attend dans une boucle bloquante que l'utilisateur entre les 4 chiffres qui représentent son identifiant de marchand. Seuls les caractères numériques du clavier sont considérés. Une fois que cela est fait, le programme affiche une DEL verte et change l'état pour `ENTER_CART` (qui est l'état par défaut pour le reste de l'exécution).

Dans cet état, le marchand est invité à entrer une commande. Pour ce faire, une deuxième MEF a été introduite. Dans celle-ci, le marchand entre d'abord un nombre représentant la quantité à acheter. Il appuie ensuite sur « F » et entre un deuxième nombre représentant le numéro du produit à acheter. Finalement, il peut appuyer sur « E » pour ajouter un autre article en suivant la même procédure ou il peut appuyer sur « C » pour terminer la commande. Notons que tous les nombres ont un maximum de 4 chiffres pour les besoins du prototype et qu'une commande est limitée à 10 items. Lors de l'entrée de nombres, seuls les caractères numériques sont considérés.

Lorsque la commande est complète, le programme revient dans la MEF principale à l'état `GET_CART_TOTAL`. Dans cet état, le terminal utilise la librairie JSON pour créer un message recevable par le serveur et l'envoie par Xbee au pont. Si tout va bien, il reçoit du serveur un message JSON qu'il decode pour avoir le total de la commande. Ce montant sera affiché sur l'écran pour le client.

Le programme change alors l'état pour `RFID`. Dans ce dernier, le programme attend de façon bloquante que le client tape sa carte sur le lecteur RFID disposé sur le devant du terminal. Une fois que la carte est lue, le programme change pour l'état `PAYMENT`. Dans cet état, le programme utilise de nouveau la librairie JSON pour créer un message recevable par le serveur et l'envoie par Xbee au pont. Si tout va bien, il reçoit du serveur un message JSON qu'il decode pour savoir si le paiement a été accepté et connaître le solde restant sur la carte. Une lumière s'affiche en conséquence (verte si tout est correct, jaune sinon) et le solde de la carte est affiché à l'utilisateur. Finalement, le programme revient dans l'état `GET_CART_TOTAL`.

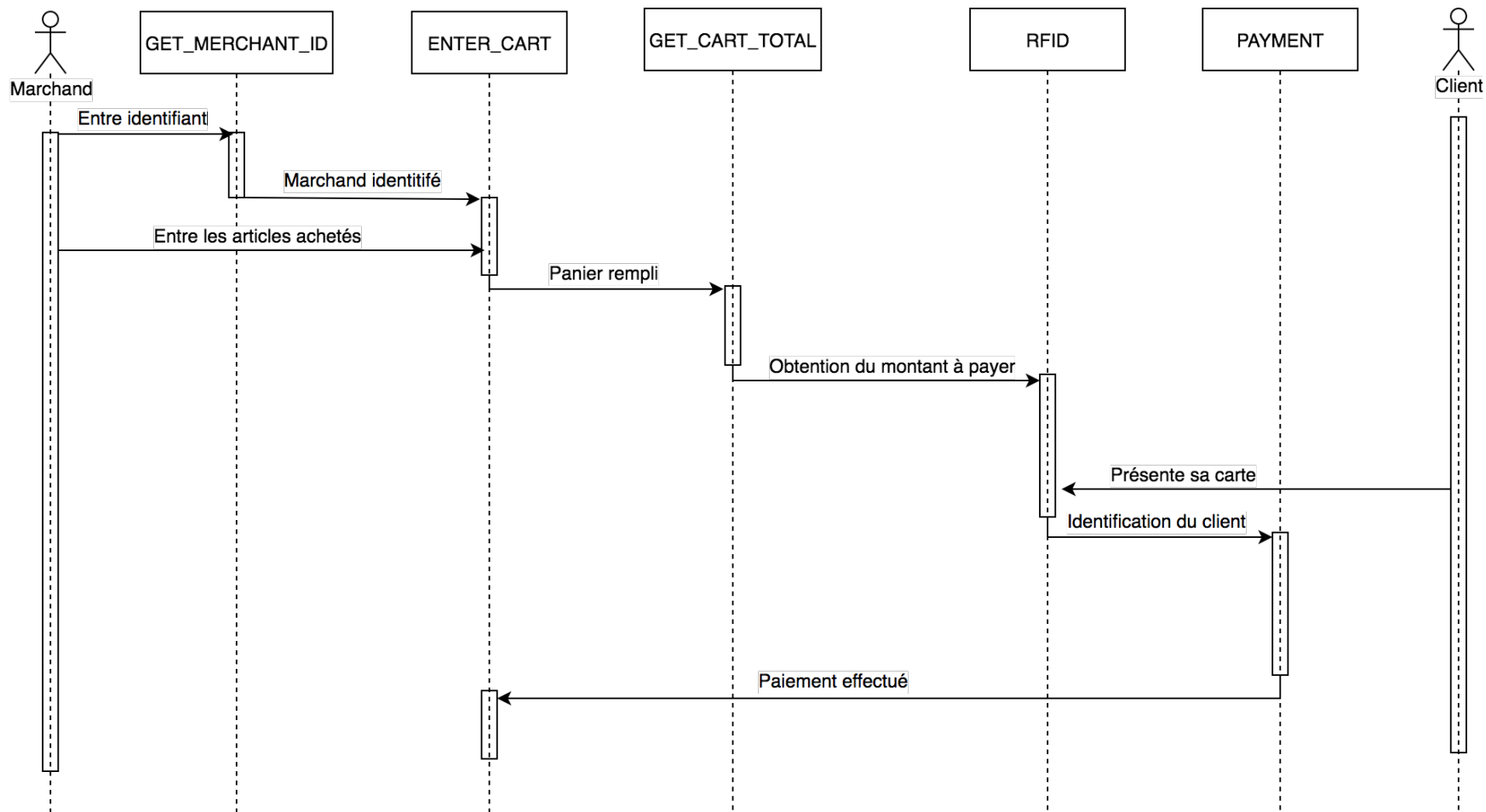


FIGURE 5-3 – Diagramme de séquence du terminal de paiement

5.3.2 Protocole

Le terminal de paiement utilise deux autres types de messages JSON sur les quatre disponibles.

Paiement Afin d'effectuer un paiement, le programme envoie les items et les quantités associées à chaque item. Le « clientId » est le numéro de la carte, et « merchantId » est le numéro que le marchand entre au démarrage du terminal.

Requête {	Réponse {
"id": NOMBRE,	"id": NOMBRE,
"method": 1,	"status": False/True,
"clientId": STRING,	"balance": NOMBRE
"merchantId": NOMBRE,	}
"items": [11, 12, 14],	
"qty": [1, 5, 2]	
}	

Total Cette méthode est semblable au paiement : elle permet de récupérer le prix de la commande selon les items que le client souhaite acheter.

Requête {	Réponse {
"id": NOMBRE,	"id": NOMBRE
"method": 4,	"total": NOMBRE
"merchantId": NOMBRE,	}
"items": [11, 12, 14],	
"qty": [1, 5, 2]	
}	

5.4 Serveur et base de données

Un serveur « node » assisté du langage javascript a été utilisé pour construire les fonctionnalités « backend » de notre projet. Le serveur est constitué de deux parties principales : un service d'api et un site web.

5.4.1 API

Le service d'api est du format REST. Il permet d'accéder, d'ajouter, de modifier ou de retirer des ressources de la base de données. Chacune de ces opérations est sans état. L'avantage d'utiliser un api REST est que toutes les plateformes intégrant une fonctionnalité de requête HTTP et de sérialisation json peuvent facilement accéder à nos services de manière standardisée et agnostique aux détails d'implémentation de notre serveur. Voici une liste des services de notre api :

- | | | |
|------------------|---------------------|----------------------|
| — Create Account | — List All Accounts | — List Items |
| — Login Account | — Create Item | — Create Transaction |
| — Get Account | — Delete Item | — List Transactions |

L'équipe a d'abord essayé d'utiliser les services ci-dessus avec le pont zigbee, cependant le micro-contrôleur du pont n'avait pas assez de mémoire pour sérialiser en JSON les informations reçues par ses nœuds, tel que mentionné précédemment. Cependant, les nœuds pouvaient le faire et le pont a donc été utilisé comme un relayeur des paquets JSON au serveur web. Toutefois, puisque le pont ne connaît pas le contenu des trames qu'il relaie, il ne peut envoyer ses requêtes qu'à un seul « endpoint » de l'api REST.

Un « endpoint » spécial nommé « zigbee/bridge » a été créé pour désérialiser le JSON relayé par le pont zigbee et exécuter les actions nécessaires. Ce service comporte 4 fonctionnalités :

- Créer une transaction (débitier et créditer les bons comptes)
- Recevoir le montant restant sur un compte client
- Ajouter de l'argent à un compte
- Afficher le total pour un achat

5.4.2 Site Web

Le site web est une interface qui permet aux marchands de voir leurs transactions effectuées, voir leurs produits, ajouter leurs produits et supprimer leurs produits. Les clients peuvent également accéder au site web pour voir leurs achats.

Le site web a été créé sur le même serveur web que celui de l'api REST. Il utilise un modèle MVC. Il possède ses propres contrôleurs, routes et vues (*views*), mais il partage ses modèles avec le service REST. Le « rendering » de l'affichage se fait du côté serveur et des requêtes ajax sont utilisées pour rafraîchir les données automatiquement sans rafraîchissement de la page.

5.4.3 Postgres avec tables

Une base de données du format *postgres* a été utilisée pour le projet, laquelle comporte 4 tables :

- Accounts
- Items
- Transactions
- LineItems

La table « Accounts » comporte les comptes des clients et des marchands, leurs informations à propos de leur CIP, leurs cartes RFID, leur PIN et leur solde. La table « Items », fait la relation entre des items et un marchand qui les possède. La table « LineItems » fait la relation entre des items, leur quantité et une transaction. La table « Transactions » fait la relation entre un client et un marchand pour une transaction.

Le plugin « sequelize » a été utilisé pour faire nos modèles de la base de données en javascript et pour faire nos migrations de base de données ainsi que des « seeds » pour la base de données.

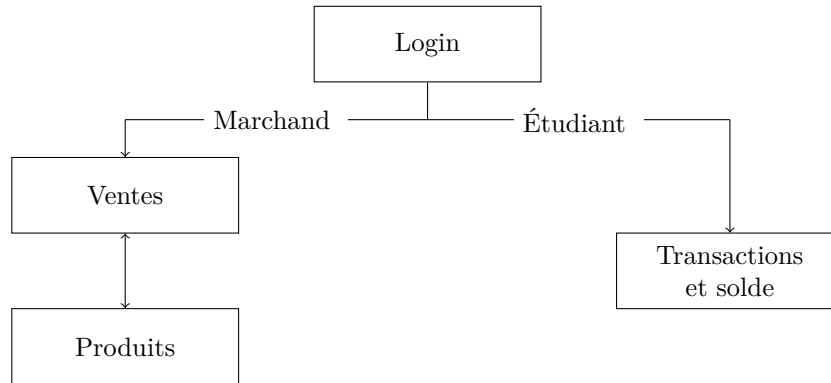


FIGURE 5-4 – Schéma des interactions avec le site web

Authentification

CIP

Soumettre

FIGURE 5-5 – Fenêtre d'authentification du site web (login)

Transactions - girp2705

[Login](#)

Article	Montant	Destinataire	Date
Beer	30.15\$	merc2020	2017/11/19

solde : 1200.00\$

FIGURE 5-6 – Transactions et solde d'un étudiant

Produits - merc2020

[Sales](#) - [Login](#)

Produits	Description	Prix	Shortcut	Action
Chicken	Awesome	7.99\$	12	supprimer
Pizza	Awesome	3.99\$	13	supprimer
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="button" value="Ajouter"/>				

FIGURE 5-7 – Liste des produits d'un marchand

Ventes - merc2020

[Products](#) - [Login](#)

Article	Unités vendues	Somme	Client	Date
Beer	3	30.15\$	girp2705	2017/11/19
<div>solde : 30.15\$</div>				

FIGURE 5-8 – Ventes d'un marchand

6 Structure du code du projet

6.1 Code embarqué

Le code de chaque composante embarquée est structuré de la même façon. À la racine du fichier, on observe les dossiers *lib* (qui contient les bibliothèques utilisées) et *src* (qui contient le code du projet). Les bibliothèques utilisées sont parfois maintenues par l'équipe, notamment pour le RFID, le clavier, le JSON et le Xbee (pour compléter la librairie officielle). Cela permet de mettre à jour le code commun de toutes les composantes facilement lorsqu'il change.

Au niveau du code source du projet, la sous-division dépend de la fonction de chaque composante, mais chaque fonction est encapsulée dans son propre fichier avec les fichiers sources et les en-têtes.

Terminal de paiement Contient des modules pour les entrées claviers, pour la communication avec le serveur par Xbee et pour gérer les LED

Terminal de recharge Contient le module pour la communication avec le serveur par Xbee

Pont Contient le module pour la communication Xbee avec les terminaux et le module pour la communication Ethernet avec le serveur

Dans tous les cas, les projets contiennent un fichier de configuration qui permet de configurer la composante facilement. Ils contiennent également un fichier debug pour activer le mode debug.

6.2 Code serveur

Au niveau du code source pour le serveur, une organisation MVC est utilisée avec le framework Express de node.js.

Voici les fichiers de configuration importants :

Package.json Liste des dépendances du projet et leurs versions

Server.js Configuration du serveur

Routes.js Routes du serveur API

Ejs_routes.js Routes de l'interface web

Docker-compose.yml Configuration de docker pour démarrer la base de données

Voici les dossiers importants pour le développement :

Models Comporte tous les modèles en relation avec la base de données

Views Comporte les vues pour le site web

Controllers Contient les contrôleurs qui font la relation entre les vues et le modèle et implémentent les API REST.

Voici les dossiers importants pour l'aide au développement de l'application :

Test Collection de tests automatisés à exécuter avant d'envoyer du code sur github pour vérifier que les nouvelles fonctionnalités n'ont rien brisés. On devrait écrire un test pour chaque fonctionnalité au fur et à mesure qu'ils sont développés

Migrations Les migrations de la base de données

Helpers Des fonctions pour aider les contrôleurs et les modèles dans leurs tâches

7 Réflexion sur les corrections à apporter à la 2^e itération

La première amélioration importante à réaliser pour la deuxième itération serait au niveau de la sécurité. En effet, pour le moment il n'y a aucune sécurité et c'est loin d'être idéal pour un système de paiement. Il faudrait commencer par chiffrer les communications Xbee pour éviter que des acteurs malveillants puissent espionner le réseau. Ensuite, il faudrait authentifier les nœuds sur le réseau afin de s'assurer qu'ils sont autorisés à effectuer des actions sur les comptes. Finalement, il faudrait aussi avoir un mécanisme d'authentification sur le site internet pour s'assurer que seul le propriétaire du compte puisse accéder à ses informations.

La deuxième amélioration qu'il serait important de réaliser est au niveau du code qui tourne sur les microcontrôleurs. Il faudrait essayer d'effectuer une gestion d'erreurs plus complète et avancée. Pour le moment, l'utilisateur a seulement une idée minimale des problèmes qui surviennent dans le système et le programme ne vérifie pas si les numéros de marchand ou de produits existent avant le paiement. Également, il faudrait que le système réessaye lorsque des erreurs de communication surviennent au lieu d'annuler la transaction. Finalement, il serait bien de faire un code un peu plus asynchrone sur les terminaux, qui sont très linéaires pour le moment, notamment pour permettre d'annuler ou modifier une transaction à n'importe quel moment dans le processus.

La troisième amélioration qui serait intéressante serait au niveau du matériel. Il faudrait d'abord incorporer le GPS tel qu'il était prévu dans le cahier des charges initial afin d'avoir une position des différents terminaux en tout temps. Ensuite, il faudrait tester de façon précise la durée de vie des terminaux sur batterie afin de pouvoir informer les clients et potentiellement les changer si nécessaire. Finalement, il faudrait essayer de réduire au minimum la consommation électrique en mettant en veille les différents appareils lorsque possible.

8 Gestion de projet

8.1 Courbe en S

Tout au long de la session, l'équipe a utilisé Airtable pour faire la division des tâches. Aussi, l'équipe a été capable de suivre facilement la planification initiale grâce à ses suivis hebdomadaires. En effet, chaque jeudi, les tâches du *sprint* finissant étaient discutées et déplacées si nécessaire. La planification du *sprint* à venir était aussi faite, ce qui permettait aux membres de l'équipe de bien prévoir leur temps et leur session de travail.

Lors des réunions hebdomadaires, on estimait la durée en heures des tâches planifiées pour le *sprint* à venir en équipe. Les membres étaient assez assidus pour entrer leurs heures travaillées au fur et à mesure. La figure 8-1 montre notre courbe en S pour toute la session.

8.2 Explication de l'allure des courbes

Plusieurs tâches ont été réalisées au début du projet jusqu'à la revue 1. Durant cette période, l'équipe a défini le projet et a fait une prévision temporelle du projet pour le restant de la session. Puisque l'équipe a bien suivi cette planification, on voit que l'avancement est linéaire après la revue 1. Il n'y a pas de grande variation des heures travaillées de semaine en semaine.

L'équipe a aussi été en mesure de ne pas prendre de grand retard durant la session. Cela s'explique à cause que nous avons tenu une gestion hebdomadaire et que s'il y avait un problème avec une tâche, on prenait action immédiatement.

Étant donné que l'équipe divisait le projet en petites tâches, l'estimation des heures a été effectuée de manière précise tout au long de la session. Le fait d'utiliser de petites tâches facilitait grandement l'estimation de la durée de la tâche, mais aussi évitait de prendre du retard. En effet, l'avancement des tâches était facile à suivre de semaine en semaine à cause qu'elles prenaient rarement plus de 4 heures à réaliser.

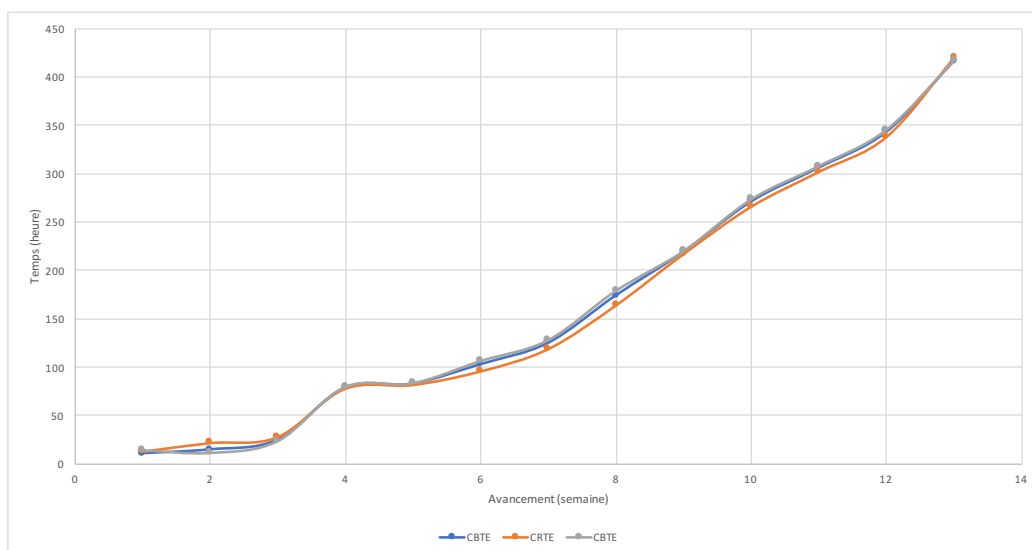


FIGURE 8-1 – Courbe en S finale montrant l'évolution temporelle du projet

9 Gestion des risques

Lors du projet, les risques ont été très bien gérés. Entre autres, nous avons réussi à diminuer la majorité des risques logiciels en explorant les multiples solutions et en vérifiant leur intégrité. Cependant, la gestion des risques a peu diminué au niveau du matériel utilisé puisque l'endommagement des composantes constitue un type de risque redondant. Alors, pour améliorer davantage la gestion des risques, on doit s'assurer que les montages physiques ne peuvent pas causer de problèmes, tels que des branchements en sens inverse. Par exemple, il existe des connecteurs à sens unique pour la connexion de batterie. Ultimement, nous pourrions faire la conception de PCB, ce qui diminuerait les risques matériels au minimum, mais cette option est peu envisageable dû à la contrainte de temps.

TABLEAU 9-6 – Grille de cotation de 1 à 10

Note F	Fréquence	Note G	Gravité	Note D	Probabilité de non-détection
10	Permanent	10	Mort d'homme	10	Aucune probabilité de détection
5	Fréquent	5	Conséquences financières et/ou matérielles	5	Possibilité de détection
1	Invraisemblable	1	Pas grave	1	Détection infaillible

TABLEAU 9-7 – Risques de gestion

Mode	Effet	Cause	F	G	D	C	Action	F	G	D	C
Airtable hors service	Aucun accès à la gestion de projet	Hors de notre contrôle	1	4	1	4	Suivre les instructions du site web	1	1	1	1
Mauvaise mise à jour des tâches	Status erroné du projet	Indiscipline des membres de l'équipe	3	2	3	18	Faire un suivi hebdomadaire	2	2	2	8
Limite de cartes sur Airtable	Aucune possibilité d'en ajouter de nouvelles	Utilisation de la version gratuite	1	1	1	1	Créer un 2 ^e Airtable	1	1	1	1
Éléments de conception manquants	La définition du projet n'est pas complète	Oubli de l'équipe	1	1	2	2	Concevoir les éléments manquants	1	1	1	1

TABLEAU 9-8 – Risques par rapport au *hardware*

Mode	Effet	Cause	F	G	D	C	Action	F	G	D	C
Mauvais branchements	Le non-fonctionnement du système	Les pièces ont été mal branchées	2	4	2	16	Vérifier les points de tests	2	2	2	8
Défectuosités	La pièce ne fonctionne pas comme prévu	La pièce est brisée	1	2	2	4	Acheter des nouvelles pièces	1	3	1	3
Manque de broches	Incapable de brancher toutes les pièces	Utilisation de toutes les broches	2	2	2	8	Mettre un multiplexeur	1	2	2	4
Manque d'événements	Les ZigBee ne reçoivent pas tous les paquets	Mauvaise communication entre les ZigBee	1	2	3	6	Gérer la réception des paquets	1	2	2	4
Température trop élevée	Incendie dans l'un des modules	Gestion des courants avec la batterie	3	2	3	18	Ventiler les modules	1	2	2	4
Court-circuit	Défectuosité des composantes électroniques	Mauvais branchements	3	2	2	12	Confectionner des branchements à sens unique	1	1	2	2
Chute fatale	Défectuosité des composantes électroniques	Choc brisant les composants	2	2	2	8	Vérifier les points de tests et changer les composantes défectueuses	1	1	2	2
Obstruction	L'utilisateur ne peut pas recharger son compte	Vandalisme. Par exemple : une gomme dans la machine	3	3	3	27	Réparation de la machine	2	3	2	12

TABLEAU 9-9 – Risques par rapport au *software*

Mode	Effet	Cause	F	G	D	C	Action	F	G	D	C
Mauvaise communication ZigBee	Les ZigBee fonctionnent mal	Problème dans le protocole de communication	2	3	3	18	Corriger les problèmes de communications (protocole)	1	3	2	6
Segmentation de mémoire	Le programme ne fonctionne pas bien	Trop grande utilisation de la mémoire Hype	2	2	3	12	Vérifier la gestion/déclaration des variables, utiliser une mémoire pool	1	2	3	6
Maintenance du code	Le code n'est pas lisible	Programmeur non soucieux du détail	2	1	1	2	Ajout de commentaires, clarification du code	1	1	1	1
Crash du programme	Le programme arrête de fonctionner	Il y a un problème majeur dans le code	1	5	2	10	Trouver les défauts du programme, mieux gérer les erreurs	1	2	2	4
Limite de performance	Le programme n'est plus en temps réel	Le CPU n'est pas assez puissant pour la tâche demandée	2	5	2	20	Développer sur un Mbed plus puissant	1	4	3	12
Json trop lourd pour le LPC1768	Le programme plante par manque de mémoire	Le LPC1768 ne dispose pas d'assez de ressources	5	3	1	15	Minimiser le json et ne pas interpréter sur le pont	1	2	3	6
Erreur traitement du serveur	Gestion des erreurs à risque	Mauvaise gestion des erreurs	1	2	2	4	Améliorer la gestion des erreurs	1	1	2	2
Mauvaise lecture des pièces de monnaie	Mauvaise détection des pièces	Erreur de lecture avec l'ADC	1	2	2	4	Améliorer la lecture des pièces	1	1	2	2

10 Améliorations futures

À la suite d'une deuxième itération, le produit devrait être assez complet pour tendre vers une première ronde de commercialisation. En effet, les principaux problèmes de sécurité auront été mitigés, l'interface utilisateur aura été améliorée et le matériel aura été testé sur une longue période.

Certaines étapes primordiales devront tout de même être réalisées. Premièrement, au niveau matériel, il est certain que les modèles des terminaux devront être recréés par ordinateur afin qu'ils puissent être produits en chaîne. Le bois sera probablement remplacé par du plastique et il sera essentiel d'engager un graphiste pour donner un meilleur « look » au produit fini. Il sera aussi bien important d'assurer la sécurité de l'argent dans le terminal de recharge en ajoutant un mécanisme de protection. Finalement, un boîtier quelconque devra être conçu pour abriter le pont.

Deuxièmement, au niveau électronique, il sera nécessaire de faire la conception des différents circuits imprimés du système afin qu'ils puissent être produits à la chaîne, testés facilement et assemblés rapidement dans les boîtiers. Cela évitera également les débranchements imprévisibles et les courts-circuits. Il sera aussi essentiel de concevoir un mécanisme de recharge des batteries. Notons que le produit devra obtenir les approbations nécessaires de la FCC¹ et de l'ISDE².

Troisièmement, le logiciel serveur devra être distribué aux clients afin qu'ils puissent avoir leur propre instance locale. Il sera probablement nécessaire d'engager des techniciens pour aider à l'installation chez les clients et à la formation du personnel.

1. FCC : *Federal Communications Commission*, une agence fédérale en charge de la régulation des télécommunications aux États-Unis.

2. ISDE : *Innovation, Sciences et Développement économique Canada*, un organisme fédéral favorisant l'accessibilité des entreprises canadiennes aux marchés mondiaux