LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

La Carotte Électronique

Halim Djerroud

révision 1.3

Note

Ce projet est à réaliser dans le cadre des SAÉs suivantes :
- SAÉ 5.01 - Concevoir, réaliser et présenter une solution technique.
- SAÉ 5.02 - Piloter un projet informatique.

Cette SAÉ, vise à obtenir des compétences permettant de comprendre le fonctionnement des systèmes de sécurité basées sur des cartes à puce et aussi améliorer les compétences de réalisation de projets informatique en groupe.

Compétences ciblées

- Réaliser des outils et applications informatiques pour les R&T.
- Utilisation du système de gestion de versions GIT et la plateforme GitHub ou GitLab pour une application porte-monnaie électronique sur cartes à puce.

Compétences requises

- Avoir des connaissances en langage C.
- Posséder une bonne maîtrise du langage Python.
- Savoir modéliser une base de données.
- Être capable de lire une documentation en anglais.
- Avoir les bases en infrastructure des réseaux
- Utilisation de git

Livrables attendus

- Étudier standard ISO 7816 et réaliser une application réseau pour piloter un porte-monnaie électronique sur cartes à puce tout en respectant les normes de cybersécurité.
- Étudier les différentes vulnérabilités des applications et apporter une solution adéquate pour sécuriser l'accès.
- Utiliser correctement le système de gestion de versions et les plateformes de gestion de versions.
- Travailler en collaboration avec les membres de l'équipe.

Table des matières

Introduction

1 Présentation

L'IUT de Vélizy souhaite développer un système de récompense pour les étudiants travailleurs et assidus connu sous le nom de code confidentiel **La Carotte Électronique**. L'idée consiste à offrir une somme d'argent aux étudiants (selon le mérite, par exemple une bonne note, participation à la vie du FabLab ...), que les étudiants peuvent dépenser en boissons chaudes (café, thé ou chocolat chaud) sur le campus.

L'IUT a commencé à mettre à la disposition des étudiants des distributeurs de boissons chaudes dans chaque bâtiment. Ces distributeurs fonctionnent à l'aide de cartes à puce. On insère sa carte dans le distributeur, si le crédit est suffisant, alors on choisi une boisson parmi ces trois : café, thé ou chocolat chaud. Une fois la boisson servit une somme de 20 centimes d'euro (0.2 euro) est débitée sur la carte.

En plus des distributeurs, l'IUT a mis en place un bureau administratif pour gérer ce projet. Les étudiants peuvent s'y rendre pour récupérer leurs cartes à puce, faire des réclamations en cas de dysfonctionnements, etc. Lorsque un étudiant se présente récupérer sa carte pour la première fois, l'agent administratif effectue une procédure de **personnalisation**, qui se déroule comme suit :

1. Vérifie l'identité de l'étudiant (carte d'étudiant)
2. Prends une nouvelle carte à puce
3. L'insère dans un lecteur de carte et la personnaliser avec les informations de l'étudiant :
   — Numéro d'étudiant
   — Nom
   — Prénom
   — La carte est initialisée à 0.00 euros
4. Remettre la nouvelle carte personnalisée à l'étudiant
5. Inscrit sur le logiciel de gestion des cartes à puce que l'étudiant en question a récupéré sa carte à puce (sa carotte)

La politique de récompense mise en place par l'établissement est la suivante :
— Tout d'abord pour inciter les étudiants à venir récupérer leurs cartes, on leur attribut d'office 1.00 (5 boissons) à toute nouvelle inscription.
— Si un enseignant souhaite attribuer un bonus à un étudiant, alors il envoie un émail au bureau administratif de gestion des cartes à puce en indiquant le numéro d'étudiant pour lequel il souhaite attribuer un bonus.
— À la réception de l'émail, l'agent administratif attribut 1.00 euros pour l'étudiant en question en utilisant le logiciel de gestion

L'IUT à aussi mit en place une borne de recharge de carte (une seule borne est disponible pour tout le campus). Les étudiants peuvent insérer leur carte à puce et effectuer les actions suivantes :
— Consulter solde qui leur reste sur la carte.
— Vérifier s'il y a des bonus qui leur sont attribués, alors dans ce cas, ils peuvent les transférer sur leur carte.
— Ils peuvent aussi recharger la carte avec leur argent, car il se peut qu'un étudiant a consommé son solde initial et il n'a pas reçu de bonus, mais souhaite tout de même utiliser sa carte.
— *Attention : le solde initial n'est pas directement disponible sur la carte. Il faut passer la borne de rechargement pour transférer le crédit initial sur la carte.*

2 Vue globale du projet

Le but de ce projet est de réaliser un système de porte-monnaie électronique à base de cartes à puce. L'idée consiste à concevoir des cartes à puce que l'utilisateur peut créditer auprès d'une autorité (borne de rechargement), puis utiliser cette carte pour des paiements auprès d'un distributeur pour effectuer des achats de boissons chaudes.

Votre rôle est de concevoir le système de cartes à puce ainsi que les logiciels de gestion, il inclut cinq parties :
- Le logiciel intégré dans la carte à puce (nom de code : Rubrovitamina)
- Le logiciel de personnalisation de carte utilisé par l'agent administratif (Lubiana)
- Le logiciel de gestion utilisé par l'agent administratif pour attribuer les récompenses (Rodelika)
- Le logiciel de la borne de recharge (Berlicum)
- La base de données (Purple Dragon)

Note : les noms attribués aux logiciels sont inspirés de noms de variétés de carottes, ils sont à titre esthétique.

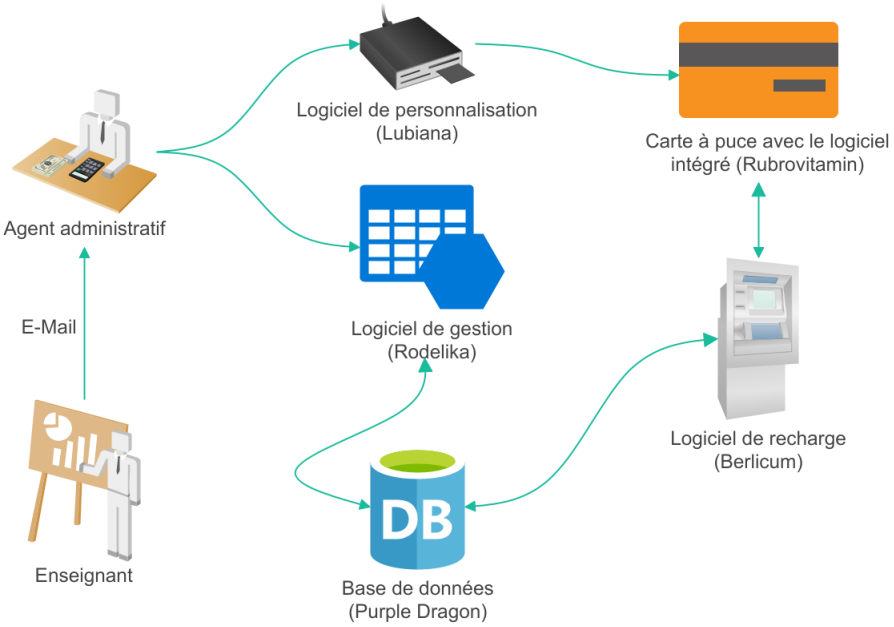

FIGURE 1 – Diagramme des relations entre les logiciels

3 Travail demandé

Votre travail consiste à développer l'ensemble logiciel de ce système (le logiciel intégré dans la carte à puce, la base de données et les applications de gestion). Pour vous aider, chaque brique de ce système vous est décrite dans une section dédiée dans la suite de ce document. Le code source partiel (dans la plupart des cas, complet) vous est donné, vous pouvez le réutiliser complètement ou partiellement, ou produire votre propre code. Si vous êtes amené à réutiliser le code donné ci-après, veuillez à ne pas faire de copier/coller à partir de ce document, mais à récrire le code manuellement ce qui vous permettra de comprendre ce que vous faites.

Pour ce projet, vous devez travailler en équipe et obligatoirement utiliser le système de gestion de versions GIT. Un responsable d'équipe sera désigné pour une période d'au maximum 2 semaines, par la suite un autre membre de l'équipe occupera cette responsabilité. Le responsable d'équipe aura pour tâches supplémentaires de gérer le dépôt Git (s'occuper des merges, pull request...) et de gérer le Kanban si vous utilisez un.

3.1 Formation des équipes

Des équipes de 2 ou 3 étudiants doivent être formées (les monômes, binômes, équipe de 4 et + ne sont pas acceptés). Il faut obligatoirement choisir un groupe et inscrire son nom et prénom sur le fichier suivant : https://lite.framacalc.org/groupes_sae_cart_ap_bwflm2wy94-a2s8

3.2 Rendu attendu

- Former son équipe et fournir un planning prévisionnel du projet à rendre la première semaine, pour le reste la date de rendu vous sera communiquée ultérieurement.
- Un lien du dépôt Github ou gitlab (les dates des commits serons vérifiées).
- Une démo fonctionnelle à montrer lors de la soutenance.

3.2 Rendu attendu

- Un document PDF d'environ 20 à 30 pages de préférence rédigé en LaTeX(pas obligatoire), essentiellement accès sur la partie IX.
- Une présentation Beamer de préférence (pas obligatoire) ou PPT à présenter lors de la soutenance.
- Démo + présentation (15min) sera présentée lors de la soutenance suivie d'une session de question/réponse (15 min).
- Examen sur table (2h00)

Première partie

La carte à puce (Rubrovitamin)

1 Vue globale

Dans cette partie, nous allons concevoir la carte à puce qui permet de stocker les bonus des étudiants. La carte doit être bien pensée. Il faut qu'elle soit personnalisable et sécurisée.

1.1 Personnalisation

La personnalisation consiste à attribuer la carte à un étudiant, c'est-à-dire écrire d'une façon persistante les informations de l'étudiant, par exemple son nom, son prénom et son numéro d'étudiant. Le processus de personnalisation est géré par l'agent administratif via un lecteur de carte à puce et un logiciel spécifique que nous allons développer dans la prochaine partie.

La personnalisation doit être uniquement accessible par personnel administratif, l'étudiant ne pourra pas changer ses informations personnelles. Pour les opérations de personnalisation, nous allons créer la classe d'instructions $0x81$ avec trois instructions :
- L'instruction $0x00$ permet de retourner la version de la carte.
- L'instruction $0x01$ permet d'entrer les informations de l'étudiant.
- L'instruction $0x02$ permet de lire les informations de l'étudiant.

Une autre classe d'instruction $0x82$ qui permet de gérer les transactions :
- L'instruction $0x01$ permet de retourner la solde de la carte.
- L'instruction $0x02$ permet de créditer la carte.
- L'instruction $0x03$ permet de débiter la carte.

CLS	Description	INS	Description	Type	P1	P2	Lc	Data	Le
0x81	Classe de personnalisation	0x00	Version de l'application	Sortie	0x00	0x00			0x04
		0x01	Entrer personnalisation	Entrée	0x00	0x00	0x07	0x43 0x61 0x72 0x72 0x6f 0x74 0x65	
		0x02	Lire persionnalisation	Sortie	0x00	0x00			0x07
0x82	Classe de gestion de paiement	0x01	Lire le sole de la carte	Sortie	0x00	0x00			0x02
		0x02	Ajouter solde 1.00 €	Entrée	0x00	0x00	0x02	0x00 0x64	
		0x03	Dépense 0.20 €	Entrée	0x00	0x00	0x02	0x00 0x14	

- Classe `0x81`
  - Instruction `0x00` :
    - Lire version : 82 00 00 00 04 → "1.00" 90 00
    - Fonction : `version();`
    - Description : Donne la version de la carte, cette dernière est directement inscrite dans le code source lors du développement de la carte :

```
#define size_ver 4
const char ver_str[size_ver] PROGMEM = "1.00";
```

    - Proposition de solution :

```
/**
  * La commande sortante associée est 82 00 00 00 04 .
  * Elle doit renvoyer 1.00 90 00.
  */

#define size_ver 4
const char ver_str[size_ver] PROGMEM = "1.00";
// émission de la version
// t est la taille de la chaîne sv
void version(){
        int i;
```

1.1 Personnalisation

```
                    // vérification de la taille
                    if (p3!=size_ver){
                        sw1=0x6c;         // taille incorrecte
                        sw2=size_ver;          // taille attendue
                        return;
                    }
                    sendbytet0(ins);         // acquittement
                    // émission des données
                    for(i=0;i<p3;i++){
                        sendbytet0(pgm_read_byte(ver_str+i));
                }
                    sw1=0x90;
            }
```

— Instruction `0x01` :
  — Introduire personnalisation : 82 01 00 00 05 "Jean" → 90 00
  — Fonction : `intro_perso();`
  — Description : Fonction de personnalisation, les données doivent être écrites dans l'eeprome. Attention `P3` ne devra pas excéder une certaine valeur.

```
        #define MAX_PERSO 32
        uint8_t ee_taille_perso EEMEM=0;
        unsigned char ee_perso[MAX_PERSO] EEMEM;
```

  — Proposition de solution :

```
        void intro_perso(){
            int i;
                unsigned char data[MAX_PERSO];
            // vérification de la taille
            if (p3>MAX_PERSO){
                        sw1=0x6c;          // P3 incorrect
                sw2=MAX_PERSO;         // sw2 contient l'information de la taille correcte
                        return;
            }
                sendbytet0(ins);         // acquitement
                for(i=0;i<p3;i++){          // boucle d'envoi du message
                    data[i]=recbytet0();
                }
                eeprom_write_block(data,ee_perso,p3);
                eeprom_write_byte(&ee_taille_perso,p3);
                sw1=0x90;
        }
```

— Instruction `0x02` :
  — Lire personnalisation : 82 02 00 00 05 → "Jean" 90 00
  — Fonction : `lire_perso();`
  — Description : Si P3 est inconnu, alors mettre `P3` à `00`. On recevra alors une erreur avec la taille attendue.
  — Proposition de solution :

```
        void lire_perso(){
                int i;
                uint8_t taille;
                taille=eeprom_read_byte(&ee_taille_perso);
                if (p3!=taille){
                        sw1=0x6c;
                        sw2=taille;
                        return;
                }
                sendbytet0(ins);
                for (i=0;i<p3;i++){
                        sendbytet0(eeprom_read_byte(data+i));
                }
                sw1=0x90;
```

1.1 Personnalisation

```
}
```

— classe `0x82` :
  — Instruction `0x01` :
    — Lire solde : 82 01 00 00 02 → xx xx 90 00
    — Fonction : `lire_solde();`
    — Description : `0€` au départ. Le solde est un `uint16_t` sauvegardé dans l'EEPROM. Il est donc compris entre 0 et 65535 centimes d'euro
    — Proposition de solution :

Le solde est un entier 16 bits (= 2 octets). Par convention, un entier 16 bits se traduit par 4 chiffres hexadécimaux qui vont de 0000 à FFFF. Il existe deux ordres de transmission des octets :
— big endian : on transmet d'abord l'octet de poids fort, puis celui de poids faible
— little endian : on transmet d'abord l'octet de poids faible, puis celui de poids fort
Les machines peuvent avoir les deux conventions mais cela dépend du modèle du processeur. Dans ce TP, on utilise la convention big endian. Si le solde est à 1€ (100 en hexadécimal donne 0x0064), on devra recevoir 00 64 90 00.

```
uint16_t solde EEMEM = 0;
void LectureSolde(){
    if(p3 != 2){
        sw1 = 0x6c ;
        sw2 = 2;
        return ;
    }
    sendbytet0(ins) ;
    uint16_t mot = eeprom_read_word(&solde) ;
    sendbytet0(mot >> 8) ; //on envoie d'abord le bit de poids fort
    sendbytet0(mot) ; //on envoie le bit de poids faible
    sw1 = 0x90 ;
}
```

  — Instruction `0x02` :
    — Créditer : 82 02 00 00 02 xx xx → 90 00 ou 61 00 (capacité maximale de rechargement dépassée)
    — Fonction : `credit();`
    — Description : instruction permettant de créditer la carte lors d'un rechargement. `61 00` si le crédit dépasse les capacités
    — Proposition de solution :

```
void credit(){
    if(p3 != 2){
        sw1 = 0x6c ;
        sw2 = 2;
        return ;
    }
    sendbytet0(ins) ;
    uint16_t ajout = ((uint16_t)recbytet0() << 8) + (uint16_t)recbytet0();
    uint16_t solde_mot = eeprom_read_word(&solde) ;
    uint16_t montant = ajout + solde_mot ;
    if(montant < ajout){ //il y a eu un debordement
        sw1 = 0x61 ;
        return ;
    }
    eeprom_write_word(&solde, montant) ;
    sw1 = 0x90 ;
}
```

  — Instruction `0x03` :
    — Débiter : 82 03 00 00 02 xx xx → 90 00 ou 61 00 (solde est insuffisant)
    — Fonction : `debit();`
    — Description : instruction permettant de débiter la carte lors d'un paiement. 61 00 si le solde est insuffisant
    — Proposition de solution :

```
void Depenser(){
    if(p3 != 2){
```

```
            sw1 = 0x6c ;
            sw2 = 2;
            return ;
        }
        sendbytet0(ins) ;
        uint16_t retrait = ((uint16_t)recbytet0()<<8) + (uint16_t)recbytet0() ;
        uint16_t solde_mot = eeprom_read_word(&solde) ;
        if(solde_mot < retrait){
            sw1 = 0x61 ; // solde insuffisant
            return ;
        }
        uint16_t montant = solde_mot - retrait ;
        eeprom_write_word(&solde, montant) ;
        sw1 = 0x90 ;
    }
```

Remarques importante

- Pas d'opérations trop lentes durant la réception des octets. Par exemple, il faut dissocier la réception des données et l'écriture de celles-ci dans l'EEPROM.
- Le solde est enregistré en uint16_t. Il faut donc convertir 2 × `uint8_t` en `uint16_t`. Pour cela, il faut faire savoir si on utilise la convention *Little endian* ou *Big endian* :

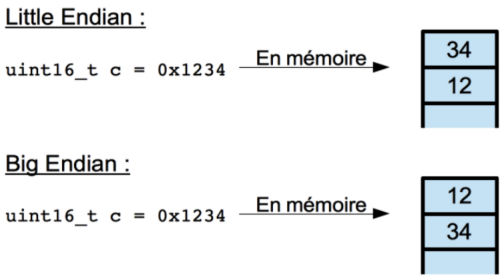

FIGURE 2 – Little Endian et Big Endian

implémentation de la boucle principale

```
 ...
// ATR
atr();
...
sw2 = 0; // pour éviter de le répéter dans toutes les commandes
// boucle de traitement des commandes
for (;;) {
  // lecture de l'entête
  cla = recbytet0();
  ins = recbytet0();
  p1 = recbytet0();
  p2 = recbytet0();
  p3 = recbytet0();
  sw2 = 0;
  switch (cla) {
  case 0x81:   // classe d'instructions de persionnalisation
    switch (ins) {
    case 0:
      version();
      break;
    case 1:
      intro_perso();
      break;
```

```
      case 2:
        lire_perso();
        break;
      default:
        sw1 = 0x6d; // code erreur ins inconnu
      }
      break;

    case 0x82:   // classe d'instructions de gestion des paiements
      switch (ins) {
      case 1:
        lire_solde();
        break;
      case 2:
        credit();
        break;
      case 3:
        debit();
        break;
      default:
        sw1 = 0x6d; // code erreur ins inconnu
      }
      break;

    default:
      sw1 = 0x6e; // code erreur classe inconnue
    }
    sendbytet0(sw1); // envoi du status word
    sendbytet0(sw2);
  }
  return 0;
}
```

1.2 Sécurité

Les cartes à puce sont une source de convoitise, elles doivent être bien sécurisées et résilientes aux attaques et dysfonctionnements. Pour un porte-monnaie électronique il faut au minimum implémenter les sécurités suivantes :

- Anti-arrachement (obligatoire pour ce projet) : si un arrachement de carte est survenu lors d'une transaction, il faut remettre la carte dans un état correcte lors de la prochaine utilisation de la carte.
- `Pin/Puk` (facultatif pour ce projet mais souhaitable) : si un étudiant brillant se fait voler sa carte alors les bonus serons consommés facilement étant donné qu'il suffit de présenter la carte au distributeur pour avoir sa boisson gratis. Pour éviter cela on souhaite ajouter un code `pin` à la carte pour rendre la tâche plus ardue aux chouravés. Un code `puk` est remit à l'étudiant lors de la remise de sa carte. Le code `puk` permet de définir un nouveau code pin en cas d'oublie. Si vous êtes amené à implémenter cette partie, toutes les opérations de manipulations de consultation de débit ou de crédit doivent être sécurisées par le code `pin`.
- Anti-rejoue (facultatif pour ce projet) : les étudiants malins peuvent essayer de bidouiller leur carte pour essayer de s'attribuer des bonus gratis.

Deuxième partie

Personnalisation (Lubiana)

La personnalisation est le processus qui permet d'attribuer une carte à puce vierge (pré-programmée) à un étudiant d'une façon unique. Dans ce projet la personnalisation se fait au niveau de l'agent administratif qui permet d'attribuer les cartes vierges (pré-programmée) aux étudiants lorsque ils se présentent pour la première fois. Le processus de personnalisation comporte deux étapes :

— Mettre les informations personnelles de l'étudiant sur la nouvelle carte
— Attribuer le solde initial de 1.00 €

Dans cette partie nous souhaitons développer un logiciel permettant à l'agent administratif d'effectuer cette tâche. La personnalisation s'effectue via un lecteur de carte à puce, l'agent doit insérer la nouvelle carte à puce dans un lecteur puis démarrer le logiciel. Nous souhaitons proposer un logiciel simple de conception et d'utilisation, pour cela, nous proposant de développer le logiciel en Python (que nous allons nommer Lubiana [1]) en ligne de commande pour qu'il reste simple à développer et interactif pour qu'il reste simple à utiliser. L'utilisation interagie avec le logiciel via le clavier en utilisant un menu interactif comme le montre l'exemple suivant :

```
-------------------------------------------
-- Logiciel de personnalisation : Lubiana --
-------------------------------------------
 1 - Afficher la version de carte
 2 - Afficher les données de la carte
 3 - Attribuer la carte
 4 - Mettre le solde initial
 5 - Consulter le solde
 6 - Quitter
Choix : _
```

Les différentes fonctionnalités :

1. **Afficher la version de carte :** permet de consulter la version de la carte à puce. Dans ce projet, la version de la carte est "1.00". Il se peut que plus tard nous développant une version "2.xx" plus sécurisée, alors l'agent administratif doit pouvoir consulter la version de carte avant de l'attribuer aux étudiants.
2. **Afficher les données de la carte** permet de savoir à quel étudiant la carte est attribuée. L'utilisateur doit être en mesure de consulter si la carte n'a pas était attribuée avant d'effectuer une nouvelle attribution.
3. **Attribuer la carte** permet d'attribuer la carte à un étudiant en saisissant les informations de l'étudiant
4. **Mettre le solde initial** permet d'ajouter un crédit de 1.00 € à la carte
5. **Consulter le solde** permet de consulter le solde de la carte. Cette opération est nécessaire avant d'ajouter un crédit à sa carte.

Utilisation de la librairie `pyscard`

On peut utiliser python pour connecter à la carte à puce utilisant la librairie *pyscard*, pour l'installer vous pouvez utiliser `pip3`. Mais avant nous allons créer un environnement virtuel python3 pour installer nos librairies. Il se peut que si vous essayez d'installer directement la librairie `pyscard`, le système vous demande d'installer au-préalable l'utilitaire `swig`.

```
$ python3 -m venv carrote
$ source ~/carrote/bin/activate
$ pip3 install swig
$ pip3 install pyscard
```

Pour vous aider, vous pouvez vous référer à la documentation officielle de la librairie qui se trouve le lien suivant : https://pyscard.sourceforge.io/user-guide.html#pyscard-user-guide

1. Variété de carotte

Afin de vous aider à réaliser ce logiciel nous vous proposons le code ci-après. Vous pouvez le réutiliser partiellement ou complètement pour vous inspirer. Attention le copier/coller n'est pas autorisé.

```
import smartcard.System as scardsys
import smartcard.util as scardutil
import smartcard.Exceptions as scardexcp

conn_reader = None

def init_smart_card():
    try:
        lst_readers = scardsys.readers()
    except scardexcp.Exceptions as e:
        print(e)
        return
    if (len(lst_readers) < 1 ):
        print(" Pas de lecteur de carte connecté !")
        exit()
    try:
        global conn_reader
        conn_reader = lst_readers[0].createConnection()
        conn_reader.connect()
        print("ATR : ", scardutil.toHexString(conn_reader.getATR()))
    except scardexcp.NoCardException as e:
        print(" Pas de carte dans le lecteur : ", e )
        exit()
    return

def print_hello_message():
    print ("-----------------------------------------------")
    print ("-- Borne de recharge : Berlicum --")
    print ("-----------------------------------------------")

def print_menu():
    print (" 1 - Afficher la version de carte ")
    print (" 2 - Afficher les données de la carte ")
    print (" 3 - Attribuer la carte ")
    print (" 4 - Mettre le solde initial ")
    print (" 5 - Consulter le solde ")
    print (" 6 - Quitter ")

def print_version():
    apdu = [0x81, 0x00, 0x00, 0x00, 0x04]
    try:
        data, sw1, sw2 = conn_reader.transmit(apdu)
    except scardexcp.Exceptions as e:
        print("Error", e)
        return
    if(sw1 != 0x90 and sw2 != 0x00):
        print ("sw1 : 0x%02X | sw2 : 0x%02X | version : erreur de lecture version" % (sw1,sw2))
    str = ""
    for e in data:
        str += chr(e)
    print ("sw1 : 0x%02X | sw2 : 0x%02X | version %s" % (sw1,sw2,str))
    return

def print_data():
    apdu = [0x81, 0x02, 0x00, 0x00, 0x05]
    data, sw1, sw2 = conn_reader.transmit(apdu)
    print ("sw1 : 0x%02X | sw2 : 0x%02X" % (sw1,sw2))
```

```
    apdu[4] = sw2
    data, sw1, sw2 = conn_reader.transmit(apdu)
    str = ""
    for e in data:
        str += chr(e)
    print ("sw1 : 0x%02X | sw2 : 0x%02X | Nom %s" % (sw1,sw2,str))
    return

def assign_card():
    apdu = [0x81, 0x01, 0x00, 0x00]
    nom = input("saisir nom : ")
    length = len(nom)
    apdu.append(length)
    for e in nom:
        apdu.append(ord(e))
    print(apdu)
    try:
        data, sw1, sw2 = conn_reader.transmit(apdu)
        print ("sw1 : 0x%02X | sw2 : 0x%02X" % (sw1,sw2))
    except scardexcp.CardConnectionException as  e:
        print("error : ", e)
    return

def assign_inital_sold():
    apdu = [0x81, 0x04, 0x00, 0x00, 0x02, 0x00, 0x64]
    try:
        data, sw1, sw2 = conn_reader.transmit(apdu)
        print ("sw1 : 0x%02X | sw2 : 0x%02X" % (sw1,sw2))
    except scardexcp.CardConnectionException as  e:
        print("error : ", e)

def read_sold():
    apdu = [0x81, 0x03, 0x00, 0x00, 0x02]
    try:
        data, sw1, sw2 = conn_reader.transmit(apdu)
        print ("sw1 : 0x%02X | sw2 : 0x%02X" % (sw1,sw2))
    except scardexcp.CardConnectionException as  e:
        print("error : ", e)
    sld = (int(data[0])*100 +  int(data[1])) / 100.00
    print ("sw1 : 0x%02X | sw2 : 0x%02X |  Solde %.2f" % (sw1,sw2,sld))

def main():
    init_smart_card()
    print_hello_message()
    while True:
        print_menu()
        cmd = int(input("Choix : "))
        if cmd == 1:
            print_version()
        elif cmd == 2:
            print_data()
        elif cmd == 3:
            assign_card()
        elif cmd == 4:
            assign_inital_sold()
        elif cmd == 5:
            read_sold()
        elif cmd == 6:
            return
```

```
        else:
            print("Commande inconnue !")
        print("\n ---\n ");
    print_menu()

if __name__ == '__main__':
    main()
```

Troisième partie

Base de données (Purple Dragon)

Pour stocker les informations des étudiants et leurs bonus, il nous faut évidemment une base de données. Nous proposons que chaque opération soit liée à un seul et unique étudiant et aussi que chaque opération soit catégorisée (Bonus, Crédit, Débit).

Nous proposons d'utiliser uniquement trois entités :

1. **Etudiant :** Cette entité représente les étudiants elle est identifiée par le numéro d'étudiant et contient uniquement les informations essentielles, à savoir le nom et le prénom.
2. **Compte :** Cette entité représente les opérations effectuées par les utilisateurs, elle est identifiée par la date de l'opération, et contient deux autres champs, le montant de l'opération et sa description.

   Étant donné que plusieurs opérations peuvent être effectuée au même moment par plusieurs utilisateurs, nous avons annoté la relation comme étant relative (R) afin d'associer le numéro de l'étudiant à la clé primaire.

   Le montant de l'opération peux être positif s'il s'agit d'un bonus ou d'un crédit et négatif s'il s'agit d'un débit.
3. **Type :** Le type de l'opération permet de catégoriser les types d'opération ce qui peux s'avérer utile s'il on souhaite faire des statistiques sur le nombre d'opérations. Pour ce projet nous allons limier les catégories à quatre types : Bonus, Bonus transféré, Crédit et Débit.

Le schémas relationnel (MCD) proposé est donné ci-après :

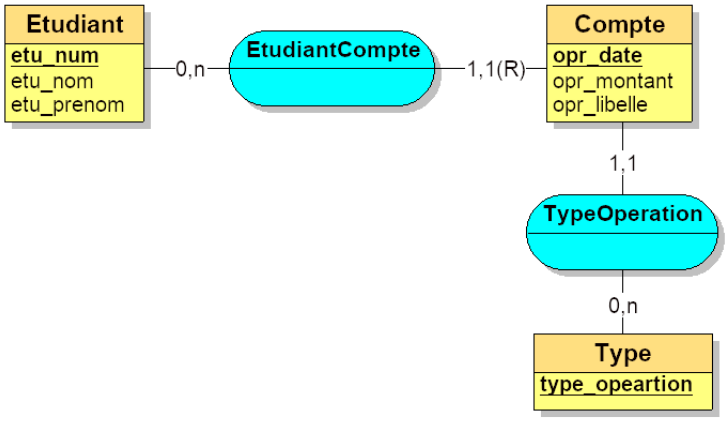

FIGURE 3 – MCD de la base de données

L'implémentation de la base de données MySql/MariaDB est donnée ci-après :

```
CREATE DATABASE IF NOT EXISTS purpledragon
  EFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
USE purpledragon;

CREATE TABLE Etudiant(
   etu_num INT AUTO_INCREMENT,
   etu_nom VARCHAR(255),
   etu_prenom VARCHAR(255),
   PRIMARY KEY(etu_num)
);

INSERT INTO etudiant (etu_num, etu_nom, etu_prenom) VALUES
(1, 'Thompson', 'Allan'),
(2, 'Castafiore', 'Bianca'),
(3, 'Lampion', 'Séraphin'),
```

```
(4, 'Da Figueira', 'Oliveira');


CREATE TABLE Type(
   type_opeartion VARCHAR(255),
   PRIMARY KEY(type_opeartion)
);

INSERT INTO type (type_opeartion)
    VALUES ('Bonus'), ('Recharge'), ('Dépense'), ('Bonus transféré');

CREATE TABLE Compte(
   etu_num INT,
   opr_date DATETIME,
   opr_montant DECIMAL(15,2) NULL DEFAULT '0.00',
   opr_libelle VARCHAR(50),
   type_opeartion VARCHAR(255) NOT NULL,
   PRIMARY KEY(etu_num, opr_date),
   FOREIGN KEY(etu_num) REFERENCES Etudiant(etu_num),
   FOREIGN KEY(type_opeartion) REFERENCES Type(type_opeartion)
);

INSERT INTO compte (etu_num, opr_date, opr_montant, opr_libelle, type_opeartion) VALUES
(1, '2023-03-09 09:15:34', '1.00', 'Initial', 'Bonus'),
(1, '2023-05-15 15:18:44', '-0.20', 'Cafe', 'Dépense'),
(1, '2023-07-11 11:14:05', '-0.20', 'Chocolat', 'Dépense'),
(1, '2023-07-24 15:20:20', '-0.20', 'Café', 'Dépense'),
(2, '2023-05-09 11:18:32', '1.00', 'Atelier Avec Dupont', 'Bonus'),
(2, '2023-05-16 08:23:32', '1.00', 'Inital', 'Bonus'),
(2, '2023-07-04 15:24:35', '1.00', 'Projet avec Pr Tournesol', 'Bonus'),
(2, '2023-07-06 15:24:35', '-0.20', 'Thé', 'Dépense'),
(2, '2023-07-26 15:23:32', '1.00', 'Chanson opéra pour Haddock', 'Bonus'),
(3, '2023-05-18 15:26:38', '1.00', 'initial', 'Bonus'),
(4, '2023-07-13 15:27:04', '1.00', 'intiial', 'Bonus'),
(4, '2023-07-21 13:10:40', '-0.20', 'Café', 'Dépense');
```

Tests : Quelques requêtes

Solde total d'un étudiant :

```
select sum(opr_montant)
from compte
where etu_num = '1'
-- Résultat 0.40
```

Solde total de tous les étudiants :

```
select etudiant.*, sum(compte.opr_montant) as sold
from etudiant, compte
where
    etudiant.etu_num = compte.etu_num
group by compte.etu_num
```

Quatrième partie

Logiciel de gestion (Rodelika)

Afin de permettre à l'agent administratif de gérer le suivi des cartes, des bonus, des débit etc. Nous avons créé une base de données *PurpleDragon*. Pour gérer cette base de données plus simplement nous souhaitons proposer une application (Rodelika [2]) interactive en ligne de commande écrite en Python.

Pour communiquer avec la base de données MySql nous allons devoir installer les librairies Python nécessaires :

```
pip3 install mysql
pip3 install mysql-connector
```

L'interface utilisateur que nous souhaitons proposer doit permettre :

- **Afficher la liste de l'ensemble des étudiants** possédant une carte.
- **Afficher le solde total** de chaque étudiant.
- **Ajouter un nouvel étudiant**, cette opération doit être effectuée systématiquement par l'agent administratif lorsque il attribue une carte à un nouvel étudiant. Mais la personnalisation de la carte se fait via le logiciel *Lubiana* développé précédemment.
- **Attribuer un bonus** à un étudiant, cette opération est effectuée par l'agent administratif lorsqu'il reçoit un émail d'un enseignant lui indiquant qu'il souhaite attribuer un bonus à un étudiant. Normalement l'enseignant doit mentionner le numéro de l'étudiant en question dans son email. Si ce l'est pas le cas alors l'agent administratif doit effectuer une recherche par nom et prénom dans la liste proposée dans la première fonctionnalité.

Un exemple de l'application minimal attendu est donné ci-après :

```
-----------------------------------
-- Logiciel de gestion : Rodlika --
-----------------------------------
 1 - Afficher la liste des étudiants
 2 - Afficher le sold des étudiants
 3 - Saisir un nouvel étudiant
 4 - Attribuer un bonus
 5 - Quitter

Choix : 1
(1, 'Thompson', 'Allan')
(2, 'Castafiore', 'Bianca')
(3, 'Lampion', 'Séraphin')
(4, 'Da Figueira', 'Oliveira')
(5, 'Roberto', 'Rastapopoulos')
(8, 'Général', 'Alcazar')


Choix : 2
(1, 'Thompson', 'Allan', Decimal('0.40'))
(2, 'Castafiore', 'Bianca', Decimal('3.80'))
(3, 'Lampion', 'Séraphin', Decimal('1.00'))
(4, 'Da Figueira', 'Oliveira', Decimal('0.80'))

Choix : 3
Nom Etudiant : Capac
Pre Etudiant : Rascar

Choix : 4
Num Etudiant : 2
```

2. Variété de carotte.

```
Commentaire : Bonus Pr Tournesol
Bonus + 1.00 euros
```

L'implémentation de cette solution est proposée ci-après :

```
import mysql.connector
cnx = mysql.connector.connect(user='*****',
                              password='******',
                              host='localhost',
                              database='purpledragon')

def print_hello_message():
    print ("-----------------------------------")
    print ("-- Logiciel de gestion : Rodlika --")
    print ("-----------------------------------")

def print_menu():
    print (" 1 - Afficher la liste des étudiants ")
    print (" 2 - Afficher le sold des étudiants ")
    print (" 3 - Saisir un nouvel étudiant ")
    print (" 4 - Attribuer un bonus ")
    print (" 5 - Quitter")

def get_list_student():
    sql="select etudiant.* from etudiant"
    cursor = cnx.cursor()
    cursor.execute(sql)
    row = cursor.fetchone()
    while row is not None:
        print(row)
        row = cursor.fetchone()

def get_list_student_with_sold():
    sql="""select etudiant.*, sum(compte.opr_montant) as sold from etudiant,
    compte where etudiant.etu_num = compte.etu_num group by compte.etu_num"""
    cursor = cnx.cursor()
    cursor.execute(sql)
    row = cursor.fetchone()
    while row is not None:
        print(row)
        row = cursor.fetchone()

def new_student():
    nom = input("Nom Etudiant : ")
    pre = input("Pre Etudiant : ")
    sql = """INSERT INTO etudiant (etu_num, etu_nom, etu_prenom) VALUES (NULL, %s,%s);"""
    val = (nom, pre)
    cursor = cnx.cursor()
    cursor.execute(sql, val)
    cnx.commit()

def add_bonus():
    num = input("Num Etudiant : ")
    com = input("Commentaire : ")
    # compléter le code
    print ("Bonus + 1.00 euros")

def main():
    # compléter le code
```

Cinquième partie

Application de gestion sur Web (Rodelika Web)

Après avoir montré notre dernière application (Rodelika) à l'agent administratif, très rapidement ce dernier nous a signaler un problème qui peut survenir lors de l'exploitation.

1 Back

```
npm init back
...
cd back
```

Installation des packages

```
npm intall extress
npm intall mysql2
```

Lancement du serveur

```
node server.js
```

2 Front

choix de vue 2

```
vue create front
# Choisir vue 2
```

Sixième partie

Borne de recharge (Berlicum)

Pour attribuer un bonus à un étudiant, l'agent administratif utilise le logiciel de gestion pour inscrire dans la base de données le montant attribué. Mais la somme n'est pas disponible sur la carte. Le rôle principal de la borne de recharge est de permettre de consulter le solde disponible sur la base de données et de le transférer sur la carte à puce.

Les bonus disponibles dans la base de données sont inscrits dans la base de données. Plus précisément dans la table `compte`, la colonne `type_operation` indique `"Bonus"` si le bonus n'a pas encore été transféré, une fois ce dernier transféré alors il la valeur de cette colonne sera changé en `"Bonus transféré"`.

Dans cette partie vous avez pour mission de développer le logiciel embarqué sur la borne de recharge *(Berlicum)*. Les opérations principales que peuvent effectuer les étudiants sont les suivantes :

- **Afficher les informations personnelles :** Cette opération permet d'afficher les informations personnelles inscrites sur la carte à puce.
- **Consulter les bonus :** Permet de consulter les bonus attribués mais pas encore transférés sur la carte.
- **Transférer les bonus :** Permet de transférer les bonus disponibles sur la carte et inscrire dans la base de données que les bonus sont bien transférées. Attention une transaction doit respecter les propriétés **ACID** [3].
- **Consulter le crédit disponible sur ma carte :** Permet simplement à un étudiant de consulter son solde disponible sur sa carte.
- **Recharger le crédit avec carte bancaire** Il se peux qu'un étudiant est épuisé ses bonus, cette opération permet de recharger le crédit de la carte à l'aide de la carte bancaire. Pour ce projet le processus de recharge avec une carte bancaire est factice, il faut simplement afficher des messages simulant la bonne transaction de la carte bancaire.

Exemple du menu principal du logiciel :

```
1 - Afficher mes informations
2 - Consulter mes bonus
3 - Transférer mes bonus sur ma carte
4 - Consulté le crédit disponible sur ma carte
5 - Recharger avec ma carte bancaire
6 - Quitter
Choix : _
```

Pour cette partie, aucun code source ne vous sera fourni. L'ensemble des fonctionnalités demandées ici, sont déjà présentées dans les sections précédentes. Vous pouvez vous inspirer du code source fourni dans les deux dernière sections.

3. Atomicité, Cohérence, Isolation et Durabilité

Septième partie

Machine à café (Lunar White)

La machine à café est le point final du système de porte-monnaie électronique. Elle permet aux étudiants de consommer des boissons chaudes (café, thé, chocolat chaud) à l'aide de leur carte à puce.

1 Fonctionnement attendu

Lorsqu'un étudiant insère sa carte à puce dans la machine, celle-ci doit permettre de sélectionner une boisson. Une fois la boisson choisie, un montant fixe de 0.20 € est débité de la carte à puce. Si le solde est insuffisant, la transaction est refusée.

Le logiciel embarqué sur la machine à café devra :
- Afficher le menu des boissons.
- Vérifier que le solde de la carte est suffisant.
- Débiter 0.20 € du solde si possible.
- Afficher un message de confirmation ou d'erreur.

L'utilisation de la carte à puce se fait via un lecteur compatible, avec les mêmes instructions que pour les autres logiciels.

2 Interface utilisateur

La machine à café fonctionne en mode autonome (pas de base de données). Un menu interactif s'affiche sur un écran (ou en mode texte pour le projet) :

```
Bienvenue à la machine à café (Lunar White)
Solde de la carte : 1.20 euros
------------------------------
Choisissez votre boisson :
1 - Café
2 - Thé
3 - Chocolat chaud
4 - Annuler
Votre choix : _
```

Après la sélection :
- Si le solde est supérieur ou égal à 0.20 €, la boisson est servie et 0.20 € sont débités.
- Sinon, un message indique que le solde est insuffisant.

3 Structure logicielle

Le logiciel peut être développé en Python avec 'pyscard' comme pour les autres modules. Exemple d'interaction pour le débit :

```python
apdu_debit = [0x82, 0x03, 0x00, 0x00, 0x02, 0x00, 0x14] # 0x0014 = 20 centimes
data, sw1, sw2 = conn_reader.transmit(apdu_debit)
if sw1 == 0x90 and sw2 == 0x00:
    print("Boisson servie. Merci !")
else:
    print("Solde insuffisant.")
```

4 Améliorations possibles

— Écrire une trace locale (log.txt) des transactions.
— Ajouter un système de verrouillage en cas de carte non authentifiée.
— Implémenter une animation en mode terminal (ASCII) ou interface graphique si souhaité, pour simuler la préparation de la boisson chaude.
— Un mode connecté à la base de données peux être ajouté mais en option, c'est à dire que les transactions ne doivent pas passer par la base de données mais par la carte à puce.

Huitième partie

Infrastructure réseau

L'architecture du système est répartie sur plusieurs machines virtuelles (VMs), chacune hébergeant un des logiciels suivants :

Nom de la machine	Logiciel installé	Rôle
-	Carte à puce	Émulation microcontrôleur / carte
`lubiana-vm`	Logiciel de personnalisation	Bureau administratif
`rodelika-vm`	Logiciel de gestion	Gestion des bonus (DB + app)
`berlicum-vm`	Borne de recharge	Transfert de bonus
`lunar-white-vm`	Machine à café	Consommation
`bd-purpledragon`	Base de données MySQL	Stockage centralisé

Chaque machine virtuelle est placée dans un VLAN distinct selon son rôle, pour simuler une segmentation réseau propre et sécurisée :

— VLAN 10 – `admin` (Lubiana, Rodelika)
— VLAN 20 – `users` (Berlicum, LunarWhite)
— VLAN 30 – `base_de_données` (PurpleDragon)

1 Travail demandé

— Déployer chaque composant sur une VM séparée.
— Affecter chaque VM au bon VLAN (avec `bridge` ou `internal network`).
— Configurer le routeur central pour permettre la communication entre les VLANs.
— Tester la connectivité (ping, telnet, accès DB, etc.).
— Sécuriser la base de données en limitant l'accès au VLAN 10 uniquement.
— Utiliser un DNS local ou '/etc/hosts' pour faciliter l'accès par nom de machine.

2 Suggestions d'outils

— **VirtualBox** ou **QEMU/KVM** pour la virtualisation.
— **PfSense** ou une VM Debian configurée en routeur pour le routage inter-VLAN.
— **GNS3/Netkit/EVE-NG** (facultatif) pour simuler routeurs et VLANs.
— **Wireshark** pour analyser le trafic réseau.

Neuvième partie

Les vulnérabilités

Les différentes applications que vous avez développées dans les sections précédentes doivent être minutieusement étudiées et testées avant leur déploiement, afin de déceler les vulnérabilités et les dysfonctionnements. Dans cette section, vous avez pour mission d'identifier les différentes vulnérabilités et de présenter et implémenter des solutions efficaces.

Vous avez pour mission de **rédiger un document** dans lequel vous allez expliquer le fonctionnement de votre projet en détail et étudier les différents problèmes, attaques, dysfonctionnements,... qui peuvent survenir. Pour chaque application, vous devez :

- Faire une présentation générale et détaillée de son fonctionnement.
- Décrire les relations avec les autres applications les modes de communication ...
- Décrire les vulnérabilités, dysfonctionnements ou attaques qui peuvent survenir.
- Proposer des solutions pour palier aux vulnérabilités contrecarrer les attaques possibles.
- En bonus (pas obligatoire pour ce projet) implémenter les mécanismes de sécurité proposés.