```
#include <Wire.h>
#include <RTClib.h>
#include <SPI.h>

//----------------------DECLARATION DES CONTANTES-------------------------
// L'objet de la minuterie
RTC_DS1307 RTC;
#define   uchar   unsigned char
#define   uint   unsigned int
//==== Broches à  câbler entre lecteur RFID / Carte Arduino UNO
// VCC sur +3.3V (alimentation)
// RST sur pin 5 (reset)
// GND sur GND (masse)
// MISO sur pin 12 (µC)
// MOSI sur pin 11 (µC)
// SCK  sur pin 13 (µC)
// NSS sur pin 10 (µC)
// IRQ  non utilisé
//=====================================================
#define MAX_LEN 16
//MF522
#define PCD_IDLE              0x00
#define PCD_AUTHENT           0x0E
#define PCD_RECEIVE           0x08
#define PCD_TRANSMIT          0x04
#define PCD_TRANSCEIVE        0x0C
#define PCD_RESETPHASE        0x0F
#define PCD_CALCCRC           0x03
//================================= Mifare_Ono
#define PICC_REQIDL           0x26
#define PICC_REQALL           0x52
```

```
#define PICC_ANTICOLL         0x93
#define PICC_SElECTTAG        0x93
#define PICC_AUTHENT1A        0x60
#define PICC_AUTHENT1B        0x61
#define PICC_READ             0x30
#define PICC_WRITE            0xA0
#define PICC_DECREMENT        0xC0
#define PICC_INCREMENT        0xC1
#define PICC_RESTORE          0xC2
#define PICC_TRANSFER         0xB0
#define PICC_HALT             0x50
//=============================== MF522
#define MI_OK                 0
#define MI_NOTAGERR           1
#define MI_ERR                2
//====================================----------MFRC522----------
//================================== Command and Status
#define    Reserved00         0x00
#define    CommandReg         0x01
#define    CommIEnReg         0x02
#define    DivlEnReg          0x03
#define    CommIrqReg         0x04
#define    DivIrqReg          0x05
#define    ErrorReg           0x06
#define    Status1Reg         0x07
#define    Status2Reg         0x08
#define    FIFODataReg        0x09
#define    FIFOLevelReg       0x0A
#define    WaterLevelReg      0x0B
#define    ControlReg         0x0C
#define    BitFramingReg      0x0D
```

```
#define    CollReg             0x0E
#define    Reserved01          0x0F
//=================================== Command
#define    Reserved10          0x10
#define    ModeReg             0x11
#define    TxModeReg           0x12
#define    RxModeReg           0x13
#define    TxControlReg        0x14
#define    TxAutoReg           0x15
#define    TxSelReg            0x16
#define    RxSelReg            0x17
#define    RxThresholdReg      0x18
#define    Reserved11          0x1A
#define    Reserved12          0x1B
#define    MifareReg           0x1C
#define    Reserved13          0x1D
#define    Reserved14          0x1E
#define    SerialSpeedReg      0x1F
//=================================== CFG
#define    Reserved20          0x20
#define    CRCResultRegM       0x21
#define    CRCResultRegL       0x22
#define    Reserved21          0x23
#define    ModWidthReg         0x24
#define    Reserved22          0x25
#define    RFCfgReg            0x26
#define    GsNReg              0x27
#define    CWGsPReg            0x28
#define    ModGsPReg           0x29
#define    TModeReg            0x2A
#define    TPrescalerReg       0x2B
```

```
#define    TReloadRegH          0x2C
#define    TReloadRegL                0x2D
#define    TCounterValueRegH          0x2E
#define    TCounterValueRegL          0x2F
//=================================== TestRegister
#define    Reserved30           0x30
#define    TestSel1Reg          0x31
#define    TestSel2Reg          0x32
#define    TestPinEnReg         0x33
#define    TestPinValueReg      0x34
#define    TestBusReg           0x35
#define    AutoTestReg          0x36
#define    VersionReg           0x37
#define    AnalogTestReg        0x38
#define    TestDAC1Reg          0x39
#define    TestDAC2Reg          0x3A
#define    TestADCReg           0x3B
#define    Reserved31           0x3C
#define    Reserved32           0x3D
#define    Reserved33           0x3E
#define    Reserved34           0x3F


//----------------------DECLARATION DE VARIABLES-------------------------
uchar serNum[5];
uchar  writeData[16]    = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100};
uchar  moneyConsume = 18 ;
uchar  moneyAdd         = 10 ;
//6Byte
uchar sectorKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
 {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
```

```
  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
};
uchar sectorNewKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff, 0x07, 0x80, 0x69, 0x19, 0x84, 0x07, 0x15, 0x76, 0x14},
  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff, 0x07, 0x80, 0x69, 0x19, 0x33, 0x07, 0x15, 0x34, 0x14},
};
int Moteur                = 3;
int buzz                  = 4;
const int NRSTPD          = 5;
int greenLed              = 6;
int redLed                = 7;
const int chipSelectPin = 10;


//----------------------DECLARATION DES BROCHES-------------------------

void setup () {
  Serial.begin(9600);
  Wire.begin();
  RTC.begin();
  if (! RTC.isrunning()) {
    Serial.println("RTC is NOT running!");
    // following line sets the RTC to the date & time this sketch was compiled
    RTC.adjust(DateTime(__DATE__, __TIME__));
  }
  Serial.begin(9600);        // RFID reader SOUT pin connected to Serial RX pin at 2400bps
  // lance la librairie SPI :
  SPI.begin();

  pinMode(chipSelectPin, OUTPUT);       // digital pin 10 en OUTPUT pour connecter au RFID
/ENABLE pin
```

```
  digitalWrite(chipSelectPin, LOW);   // Activer le lecteur RFID
  pinMode(NRSTPD, OUTPUT);            // digital pin 5 , Not Reset and Power-down
  digitalWrite(NRSTPD, HIGH);

  MFRC522_Init();  //Initialise le lecteur RFID
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(buzz, OUTPUT);
  pinMode(Moteur, OUTPUT);
  RTC.adjust(DateTime(__DATE__, __TIME__));
}

//----------------------PROGRAMME PRINCIPALE-------------------------

void loop()
{
  uchar i, tmp;
  uchar status;
  uchar str[MAX_LEN];
  uchar RC_size;
  uchar blockAddr;
  String mynum = "";
  //status
  status = MFRC522_Request(PICC_REQIDL, str);
  if (status == MI_OK)
  {
    Serial.println("OK!!!");
  }
  status = MFRC522_Anticoll(str);
  memcpy(serNum, str, 5);
  if (status == MI_OK)
```

```
  {
   Affich_Card();                          //Fonction Affichage du numÃ©ro de carte
   if (serNum[0] == 100) {                    /*Regarder si on a reconnu une carte dÃ©terminÃ©e*/
    Pass_Ok();                             //Fonction si le pass est bon
   } else if (serNum[0] != 100) {
    Pass_Error();                           //Fonction si le pass n'est pas reconnu
   }
   delay(1000);
  }
  MFRC522_Halt();
}

//----------------------DEFINITION DE FONCTIONS-------------------------

//======= Write_MFRC5200  addr-- ; val--  ======================
void Write_MFRC522(uchar addr, uchar val)
{
  digitalWrite(chipSelectPin, LOW);

  //ï¼š0XXXXXX0
  SPI.transfer((addr << 1) & 0x7E);
  SPI.transfer(val);

  digitalWrite(chipSelectPin, HIGH);
}



//======== Read _MFRC522 : addr---  ==========================
uchar Read_MFRC522(uchar addr)
{
  uchar val;
```

```
  digitalWrite (chipSelectPin, LOW);

  //ï¼š1XXXXXX0
  SPI.transfer (((addr << 1) & 0x7E) | 0x80);
  val = SPI.transfer(0x00);

  digitalWrite (chipSelectPin, HIGH);

  return val;
}

//=========== SetBitMask  RC522ï¼šreg-- ;mask--  ====================
void SetBitMask(uchar reg, uchar mask)
{
  uchar tmp;
  tmp = Read_MFRC522(reg);
  Write_MFRC522(reg, tmp | mask);  // set bit mask
}


// === ClearBitMask RC522 ï¼šreg--;mask--  ==============
void ClearBitMask(uchar reg, uchar mask)
{
  uchar tmp;
  tmp = Read_MFRC522(reg);
  Write_MFRC522(reg, tmp & (~mask));  // clear bit mask
}


// ========= AntennaOn ======================================
```

```
void AntennaOn(void)
{
  uchar temp;

  temp = Read_MFRC522(TxControlReg);
  if (!(temp & 0x03))
  {
    SetBitMask(TxControlReg, 0x03);
  }
}


// ======== AntennaOff ==================================
void AntennaOff(void)
{
  ClearBitMask(TxControlReg, 0x03);
}


//======== ResetMFRC522 ================================
void MFRC522_Reset(void)
{
  Write_MFRC522(CommandReg, PCD_RESETPHASE);
}


//====== InitMFRC522 ================================
void MFRC522_Init(void)
{
  digitalWrite(NRSTPD, HIGH);
```

```
  MFRC522_Reset();

  //Timer: TPrescaler*TreloadVal/6.78MHz = 24ms
  Write_MFRC522(TModeReg, 0x8D);      //Tauto=1; f(Timer) = 6.78MHz/TPreScaler
  Write_MFRC522(TPrescalerReg, 0x3E);   //TModeReg[3..0] + TPrescalerReg
  Write_MFRC522(TReloadRegL, 30);
  Write_MFRC522(TReloadRegH, 0);

  Write_MFRC522(TxAutoReg, 0x40);      //100%ASK
  Write_MFRC522(ModeReg, 0x3D);      //CRCåˆ•å§‹å€¼0x6363   ???

  //ClearBitMask(Status2Reg, 0x08);      //MFCrypto1On=0
  //Write_MFRC522(RxSelReg, 0x86);      //RxWait = RxSelReg[5..0]
  //Write_MFRC522(RFCfgReg, 0x7F);        //RxGain = 48dB

  AntennaOn();
}



/* ==== MFRC522_Request ï¼šreqModeï¼Œ ====================
          TagType--
             0x4400 = Mifare_UltraLight
             0x0400 = Mifare_One(S50)
             0x0200 = Mifare_One(S70)
             0x0800 = Mifare_Pro(X)
             0x4403 = Mifare_DESFire
    MI_OK
*/
uchar MFRC522_Request(uchar reqMode, uchar *TagType)
{
  uchar status;
```

```
  uint backBits;

  Write_MFRC522(BitFramingReg, 0x07);      //TxLastBists = BitFramingReg[2..0]   ???

  TagType[0] = reqMode;
  status = MFRC522_ToCard(PCD_TRANSCEIVE, TagType, 1, TagType, &backBits);

  if ((status != MI_OK) || (backBits != 0x10))
  {
    status = MI_ERR;
  }

  return status;
}



/* MFRC522_ToCard RC522 ISO14443 : command--MF522
          sendData--RC522,
          sendLen--
          backData--ï¼Œ
          backLen--
   MI_OK
*/
uchar MFRC522_ToCard(uchar command, uchar *sendData, uchar sendLen, uchar *backData, uint
*backLen)
{
  uchar status = MI_ERR;
  uchar irqEn = 0x00;
  uchar waitIRq = 0x00;
  uchar lastBits;
  uchar n;
```

```
uint i;

switch (command)
{
 case PCD_AUTHENT:
   {
    irqEn = 0x12;
    waitIRq = 0x10;
    break;
   }
 case PCD_TRANSCEIVE:   //FIFO
   {
    irqEn = 0x77;
    waitIRq = 0x30;
    break;
   }
 default:
   break;
}

Write_MFRC522(CommIEnReg, irqEn | 0x80);
ClearBitMask(CommIrqReg, 0x80);
SetBitMask(FIFOLevelReg, 0x80);        //FlushBuffer=1, FIFO

Write_MFRC522(CommandReg, PCD_IDLE);   //NO action;

//FIFO
for (i = 0; i < sendLen; i++)
{
 Write_MFRC522(FIFODataReg, sendData[i]);
}
```

```
//
Write_MFRC522(CommandReg, command);
if (command == PCD_TRANSCEIVE)
{
  SetBitMask(BitFramingReg, 0x80);      //StartSend=1,transmission of data starts
}

//
i = 2000;   //25ms   ???
do
{
  //CommIrqReg[7..0]
  //Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq
  n = Read_MFRC522(CommIrqReg);
  i--;
}
while ((i != 0) && !(n & 0x01) && !(n & waitIRq));

ClearBitMask(BitFramingReg, 0x80);        //StartSend=0

if (i != 0)
{
  if (!(Read_MFRC522(ErrorReg) & 0x1B))  //BufferOvfl Collerr CRCErr ProtecolErr
  {
    status = MI_OK;
    if (n & irqEn & 0x01)
    {
      status = MI_NOTAGERR;        //??
    }
```

```
 if (command == PCD_TRANSCEIVE)
 {
   n = Read_MFRC522(FIFOLevelReg);
   lastBits = Read_MFRC522(ControlReg) & 0x07;
   if (lastBits)
   {
     *backLen = (n - 1) * 8 + lastBits;
   }
   else
   {
     *backLen = n * 8;
   }

   if (n == 0)
   {
     n = 1;
   }
   if (n > MAX_LEN)
   {
     n = MAX_LEN;
   }

   //FIFO
   for (i = 0; i < n; i++)
   {
     backData[i] = Read_MFRC522(FIFODataReg);
   }
 }
}
else
{
```

```
    status = MI_ERR;
   }

  }

  //SetBitMask(ControlReg,0x80);          //timer stops
  //Write_MFRC522(CommandReg, PCD_IDLE);

  return status;
}


//=== MFRC522_Anticoll serNum-- ==============
uchar MFRC522_Anticoll(uchar *serNum)
{
  uchar status;
  uchar i;
  uchar serNumCheck = 0;
  uint unLen;


  //ClearBitMask(Status2Reg, 0x08);      //TempSensclear
  //ClearBitMask(CollReg,0x80);        //ValuesAfterColl
  Write_MFRC522(BitFramingReg, 0x00);      //TxLastBists = BitFramingReg[2..0]

  serNum[0] = PICC_ANTICOLL;
  serNum[1] = 0x20;
  status = MFRC522_ToCard(PCD_TRANSCEIVE, serNum, 2, serNum, &unLen);

  if (status == MI_OK)
  {
```

```
    //
    for (i = 0; i < 4; i++)
    {
      serNumCheck ^= serNum[i];
    }
    if (serNumCheck != serNum[i])
    {
      status = MI_ERR;
    }
  }

  //SetBitMask(CollReg, 0x80);      //ValuesAfterColl=1

  return status;
}


// CalulateCRC  MF522 CRC ï¼špIndata--ï¼Œlen--ï¼ŒpOutData--
void CalulateCRC(uchar *pIndata, uchar len, uchar *pOutData)
{
  uchar i, n;

  ClearBitMask(DivIrqReg, 0x04);        //CRCIrq = 0
  SetBitMask(FIFOLevelReg, 0x80);       //FIFO
  //Write_MFRC522(CommandReg, PCD_IDLE);

  //FIFO
  for (i = 0; i < len; i++)
  {
    Write_MFRC522(FIFODataReg, *(pIndata + i));
  }
```

```
  Write_MFRC522(CommandReg, PCD_CALCCRC);

  //CRC
  i = 0xFF;
  do
  {
    n = Read_MFRC522(DivIrqReg);
    i--;
  }
  while ((i != 0) && !(n & 0x04));     //CRCIrq = 1

  //CRC
  pOutData[0] = Read_MFRC522(CRCResultRegL);
  pOutData[1] = Read_MFRC522(CRCResultRegM);
}


// === MFRC522_SelectTagï¼šserNum--
uchar MFRC522_SelectTag(uchar *serNum)
{
  uchar i;
  uchar status;
  uchar size;
  uint recvBits;
  uchar buffer[9];

  //ClearBitMask(Status2Reg, 0x08);        //MFCrypto1On=0

  buffer[0] = PICC_SElECTTAG;
  buffer[1] = 0x70;
  for (i = 0; i < 5; i++)
```

```
  {
    buffer[i + 2] = *(serNum + i);
  }
  CalulateCRC(buffer, 7, &buffer[7]);      //??
  status = MFRC522_ToCard(PCD_TRANSCEIVE, buffer, 9, buffer, &recvBits);

  if ((status == MI_OK) && (recvBits == 0x18))
  {
    size = buffer[0];
  }
  else
  {
    size = 0;
  }

  return size;
}



/* MFRC522_Auth

  ï¼šauthMode--
          0x60 = éªŒè¯Aå¯†é’¥
          0x61 = éªŒè¯Bå¯†é’¥
          BlockAddr--å—åœ°å€
          Sectorkey--æ‰‡åŒºå¯†ç 
          serNum--å¡ç‰‡åºåˆ—å·ï¼Œ4å-—èŠ,
    MI_OK
*/
uchar MFRC522_Auth(uchar authMode, uchar BlockAddr, uchar *Sectorkey, uchar *serNum)
{
```

```
  uchar status;
  uint recvBits;
  uchar i;
  uchar buff[12];

  //
  buff[0] = authMode;
  buff[1] = BlockAddr;
  for (i = 0; i < 6; i++)
  {
    buff[i + 2] = *(Sectorkey + i);
  }
  for (i = 0; i < 4; i++)
  {
    buff[i + 8] = *(serNum + i);
  }
  status = MFRC522_ToCard(PCD_AUTHENT, buff, 12, buff, &recvBits);

  if ((status != MI_OK) || (!(Read_MFRC522(Status2Reg) & 0x08)))
  {
    status = MI_ERR;
  }

  return status;
}


// ======= MFRC522_Read ï¼šblockAddr--;recvData--
uchar MFRC522_Read(uchar blockAddr, uchar *recvData)
{
  uchar status;
```

```
  uint unLen;

  recvData[0] = PICC_READ;
  recvData[1] = blockAddr;
  CalulateCRC(recvData, 2, &recvData[2]);
  status = MFRC522_ToCard(PCD_TRANSCEIVE, recvData, 4, recvData, &unLen);

  if ((status != MI_OK) || (unLen != 0x90))
  {
    status = MI_ERR;
  }

  return status;
}


// ====MFRC522_Write ï¼šblockAddr--;writeData-- 16
uchar MFRC522_Write(uchar blockAddr, uchar *writeData)
{
  uchar status;
  uint recvBits;
  uchar i;
  uchar buff[18];

  buff[0] = PICC_WRITE;
  buff[1] = blockAddr;
  CalulateCRC(buff, 2, &buff[2]);
  status = MFRC522_ToCard(PCD_TRANSCEIVE, buff, 4, buff, &recvBits);

  if ((status != MI_OK) || (recvBits != 4) || ((buff[0] & 0x0F) != 0x0A))
  {
```

```
    status = MI_ERR;
  }

  if (status == MI_OK)
  {
    for (i = 0; i < 16; i++)  //FIFO 16Byte
    {
      buff[i] = *(writeData + i);
    }
    CalulateCRC(buff, 16, &buff[16]);
    status = MFRC522_ToCard(PCD_TRANSCEIVE, buff, 18, buff, &recvBits);

    if ((status != MI_OK) || (recvBits != 4) || ((buff[0] & 0x0F) != 0x0A))
    {
      status = MI_ERR;
    }
  }

  return status;
}


// === MFRC522_Halt ==============
void MFRC522_Halt(void)
{
  uchar status;
  uint unLen;
  uchar buff[4];

  buff[0] = PICC_HALT;
  buff[1] = 0;
```

```
  CalulateCRC(buff, 2, &buff[2]);

  status = MFRC522_ToCard(PCD_TRANSCEIVE, buff, 4, buff, &unLen);
}
// ===Valid_Pass==================
void Pass_Ok(void)
{
  int i = 1;
  card();
  for (i; i <= 4; i++) {
    digitalWrite(greenLed, HIGH);   // fixe la led comme alummÃ©
    digitalWrite(buzz, HIGH);     // fixe le buzzer comme actif
    delay(500);
    digitalWrite(greenLed, LOW);   // fixe la led comme alummÃ©
    digitalWrite(buzz, LOW);     // fixe le buzzer comme actif
    delay(250);
  }
  Affich_Date();
  digitalWrite(greenLed, HIGH);   // fixe la led comme alummÃ©
  delay(500);
  digitalWrite(greenLed, LOW);
  Out_Work();
}
void Pass_Error(void)
{
  int i = 1;
  card();
  for (i; i <= 10; i++) {
    digitalWrite(redLed, HIGH);   // fixe la led comme alummÃ©
    digitalWrite(buzz, HIGH);     // fixe le buzzer comme actif
    delay(250);
```

```
   digitalWrite(redLed, LOW);   // fixe la led comme alummÃ©
   digitalWrite(buzz, LOW);     // fixe le buzzer comme actif
   delay(150);
  }
  Affich_Date();
  digitalWrite(redLed, HIGH);
  delay(500);
  digitalWrite(redLed, LOW);
}
void Affich_Card(void)
{
  Serial.println("\nCarte detectee adresse : ");
  Serial.print(serNum[0], HEX);
  Serial.print(": ");
  Serial.print(serNum[1], HEX);
  Serial.print(": ");
  Serial.print(serNum[2], HEX);
  Serial.print(": ");
  Serial.print(serNum[3], HEX);
  Serial.print(": ");
  Serial.print(serNum[4], HEX);
  Serial.println(" ");
}
void Affich_Date(void)
{
  DateTime now = RTC.now();
  Serial.print(now.day(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.year(), DEC);
```

```
  Serial.print(' ');
  Serial.print("=>");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.println();
  Serial.println();
  delay(1000);
  if (serNum[1] == 00000100){
    Serial.print("Monsieur TONET\n");
  }
  if (serNum[1] == 101001){
    Serial.print("Unknow\n");
  }
}
void Out_Work (void)
{
  int i = 1;
  delay(119999);
  for (i; i <= 10; i++) {
    digitalWrite (redLed, HIGH);   // fixe la led comme alummÃ©
    digitalWrite (buzz, HIGH);     // fixe le buzzer comme actif
    delay (50);
    digitalWrite (redLed, LOW);   // fixe la led comme alummÃ©
    digitalWrite (buzz, LOW);     // fixe le buzzer comme actif
    delay(50);
  }
  card ();
  Serial.println (" doit partir!!!\n ");
  Serial.print ("Il est : ");
  Affich_Hour ();
```

```
}
void Affich_Hour(void)
{
  DateTime now = RTC.now();
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.println();
  Serial.println();
  delay(1000);
}
void card (void)
{
  if (serNum[1] == 00000100){
    Serial.print("Monsieur TONET\n");
  }
  if (serNum[1] == 101001){
    Serial.print("Unknow\n");
  }
}
```

Le code permet :

- de reconnaitre une carte(ou badge magnétique)

- de validé grâce au moniteur série, aux leds et a un buzzer

- de signalé quand la personne intéressé doit partir après son temps de travail

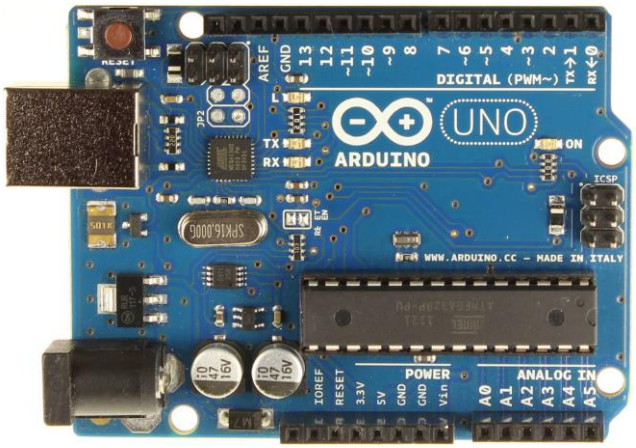
Je me suis servi

- **D'un microcontrôleur « Arduino Uno »**
  - *Je n'ai que ça chez moi*
- **D'un module « RFID RC-522 »**
  - *détection de cartes et badge.*
- **Un module « Tiny RTC DS1307 »**

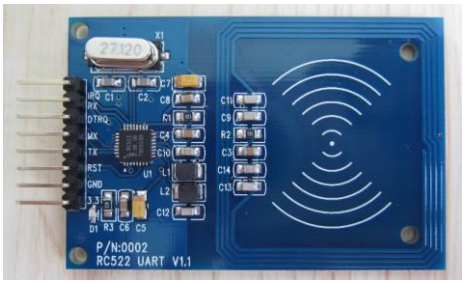
- *avoir l'heure réelle à afficher.*

- **De deux leds (1 verte et 1 rouge)**
  - *signalé si c'est validé ou pas*
- **D'un buzzer**
  - *avoir une signalisation sonore en plus*

µcontroller Arduino:



MODULE RFID RC-522:



MODULE Tiny RTC DS1307:

LEDS :

BUZZER :