

```

/*
*****
*****
Projet Centrale domotique DomoMaison 2017 pour carte Arduino Mega 2560 et carte Ethernet (W5100)
Version 1.70 du 06/09/2017
Copyright 2017 PRIN Jean-Luc
Ce programme fonctionne avec le protocole Websockets (HTML5).
*/

// -----
// ***** ENTETE DECLARATRICE *****
// -----
// Dans cette section, sont déclarées les librairies, les constantes et les variables
// ----- Section librairie et Définitions -----
// --- Inclusion des librairies de base nécessaire au système ---
#include <Arduino.h>
#include <Wire.h> // librairie I2C
#include <DS1307RTC.h> // librairie pour RTC DS1307
#include <LiquidCrystal.h> // librairie pour afficheur LCD
#include <SPI.h> // librairie SPI - obligatoire avec librairie Ethernet
#include <Ethernet.h> // librairie Ethernet
#include <EthernetUdp.h>
#include <Time.h> // librairie pour la gestion du temps
#include <SD.h> // librairie pour gestion de la carte SD
#include <WebSocket.h> // librairie nécessaire pour le protocole WebSocket
#include <limits.h> // Informations sur les constantes de compilation
#include <EEPROM.h> // librairie pour lr stockage de données en mémoire EEPROM interne
#include <DHT.h> // librairie pour les sondes de températures DHT22

// ----- fichier de configuration du logiciel -----
#include "Config_DomoMaison.h" // librairie contenant les variables spécifiques à chaque installation

// ----- Section Constantes et variables globales -----
// format des caractères de gestion de la trame WebSocket (CSV)
#define STX '(' // Caractère de début de trame pour les messages
#define ETX ')' // Caractère de fin de trame pour les messages
#define Separator ';' // Caractère Point-virgule séparateur pour les messages
// Longueur maximum de la trame utilisée par les WebSockets (ne pas modifier)..
#define MAX_FRAME_LENGTH 64
char EvtServer[126]; // chaine d'émission de message du serveur vers le client (126 caractères maximum si non
extend)
char NumCard[] = NUM_BOARD; // chaine représentant l'identifiant de la carte
byte PageWeb = 0; // numéro de la page Web en cours

// valeur courante du compteur pour le RTC
int Compteur_Rtc;
// valeur courante du compteur de période NTP
byte Compteur_Ntp;

//compteurs de ticks à la milliseconde pour synchronisation horloge
unsigned long Tick_Old = 0;
unsigned long Tick_New = 0;
unsigned long Epoch; // Heure Unix
unsigned long Epoch_Local; // Heure en local de la centrale

//compteur de tick pour l'envoi d'information entrées analogiques, compteurs d'impulsions et sondes température.
unsigned long Tick_Ana = 0;

//Indicateur de fonctionnement de la procédure d'envoi forcée des informations analogiques,des compteurs d'impulsion et des
sondes de température.
byte Indic_Init_Ana = 1;

// Variables pour la gestion dt temps
byte Hours = 0;
byte Mins = 0;
byte Sec = 0;
byte DateJour = 0;
byte JourSemaine = 0;
byte Mois = 0;
int Annee = 0;

byte JourCourant = 0; // déclaration et initialisation de la variable jour de la semaine courant
byte Dls = 0; // Day Light Saving : heure d'hiver DLS = 0, d'heure d'été DLS = 1

int TempsMn = 0; // déclaration et initialisation variable temps courant en minutes
int TempsLDR = 0; // déclaration et initialisation variable temps détection pénombre LDR
int TempsLED = 0; // temps de clignotement LED
const int TEMPS_LED_NORMAL = 1000; // durée du clignotement LED en ms (fonctionnement normal)
const int TEMPS_LED_PANNE = 200; // durée du clignotement LED en ms (fonctionnement anomalie)

// Initialisation de l'afficheur LCD
const byte NUM_ROWS = 4; // nombre de lignes de l'afficheur
const byte NUM_COLS = 20; // nombre de colonnes de l'afficheur
LiquidCrystal lcd(LCD_RS, LCD_ENABLE, LCD_D4, LCD_D5, LCD_D6, LCD_D7); // mode 4 bits - RW non connectée
int Compteur_Lcd; //valeur courante du compteur de période
LCD.

// Variables pour le contrôle d'accès
const char CODE_CARTE[] = CODE_ACCES;
int Compteur_Deconnexion_Nb_Periode = 0; // Compteur de période avant la déconnexion automatique
byte Indic_Acces = 0; // Indicateur de l'autorisation d'accès aux informations de la carte
byte Indic_Interdit = 0; // Indicateur d'émission de message d'interdiction d'accès

// Variables pour la carte SD
boolean CarteSD = false;
boolean TestSD = false; // variable mémorisation résultat renvoyé par fonction test (carte SD)

// Initialisation d'un serveur websocket

```

[illegible]

```

pinMode(53,OUTPUT); // laisser la broche SS en sortie - obligatoire avec librairie SD
digitalWrite(53,HIGH); // nécessaire pour SD avec Mega

digitalWrite(PIN_BLLCD,1); // Back Light Allumage de l'écran LCD
TempsLED = TEMPS_LED_NORMAL; // fonctionnement normal et clignotement normal de la LED
// ----- Fin de la configuration des broches Entrée/Sortie du programme système -----
-----

// ----- Début de la configuration des broches Entrée/Sortie de la gestion volets, portail, porte de garage ----
-----
pinMode(PIN_VOL_GAUCHE,OUTPUT);
pinMode(PIN_VOL_MILIEU,OUTPUT);
pinMode(PIN_VOL_DROIT,OUTPUT);
pinMode(PIN_VOL_HAUT,OUTPUT);
pinMode(PIN_VOL_STOP,OUTPUT);
pinMode(PIN_VOL_BAS,OUTPUT);
pinMode(PIN_PORTAIL,OUTPUT);
pinMode(PIN_LIBRE_1,OUTPUT);
pinMode(PIN_GARAGE_1,OUTPUT);
pinMode(PIN_GARAGE_2,OUTPUT);
// ----- Fin de la configuration des broches Entrée/Sortie de la gestion volets, portail, porte de garage -----
-----

// ----- Début de la configuration des broches Entrée/Sortie de la gestion des températures -----
-----
pinMode(PIN_DHT_1,INPUT);
pinMode(PIN_DHT_2,INPUT);
pinMode(PIN_DHT_3,INPUT);
pinMode(PIN_DHT_4,INPUT);
pinMode(PIN_DHT_5,INPUT);
// ----- Fin de la configuration des broches Entrée/Sortie de la gestion des températures -----
-----

// ----- Début de la partie système de base du programme -----
-----
// Initilisation de l'afficheur LCD
lcd.begin(NUM_COLS,NUM_ROWS); // afficheur 20 colonnes, 4 lignes
lcd.clear(); // efface LCD
lcd.setCursor(0,0); // positionne le curseur colonne 1, ligne 1
lcd.print("DOMOMAISON 2017");
lcd.setCursor(0,1);
lcd.print("Version 1.70");
delay(1000);

// initialisation du compteur pour le rafraichissement de l'afficheur
Compteur_Lcd = 0;

// Initialisation de la carte SD (à supprimer si vous ne mettez pas de carte)
testCarteSD(); // Test de la présence d'une carte SD et de son bon fonctionnement

// Initialisation des compteur RTC et NTP
Compteur_Rtc = 0;
Compteur_Ntp = 0;

// Initialisation horloge interne de la carte
Tick_Old = 0L;
Tick_New = 0L;

lectureRTC(); // Lecture de la date et de l'heure sur le module DS1307RTC et initialisation des variables
Temps

// Mise à jour de l'horloge du DS1307 à partir d'un site NTP
if (MISE_JOUR_NTP == 1) { // Mode mise à jour horloge par NTP demandée
  lcd.setCursor(8,3); // positionne curseur colonne 9, ligne 4
  lcd.print("NTP en cours");
  #ifdef DEBUG
    Serial.println(F("NTP en cours"));
  #endif
  Ethernet.begin(MAC);
  if (Ethernet.begin(MAC) == 0) { // Si la connexion web échoue
    lcd.setCursor(8,3); // positionne curseur colonne 9, ligne 4
    lcd.print("Pb NTP ");
  }
  else
  {
    calcul_ntp_time(ntp()); // on s'occupe d'abord de récupérer l'heure via serveur NTP +
affichage
    lectureRTC(); // on relie le RTC pour le calcul des dates et heures
  }
}

JourCourant = JourSemaine; // mémorisation du jour de la semaine en cours
TempsMn = Hours * 60 + Mins; // calcul du temps courant en minutes

affiDateHeure(2); // affichage de la date et de l'heure en ligne 3

// Initialisation de la carte Ethernet
Ethernet.begin(MAC, IP_LOCAL); // avec ip fixe
//Ethernet.begin(MAC); // avec ip DHCP
delay(1000);
#ifdef DEBUG
  Serial.print(F("Shield Ethernet OK : L'adresse IP du shield Ethernet est : " ));
  Serial.println(Ethernet.localIP());
#endif
wsServer.registerConnectCallback(&onConnect);
wsServer.registerDataCallback(&onData);
wsServer.registerDisconnectCallback(&onDisconnect);
wsServer.begin();
#ifdef DEBUG
  Serial.println(F("Serveur Ethernet OK : Ecoute sur port 80 (http)"));
#endif

// Initialisation des variables d'accès

```

```

Indic_Acces = 0; // Indicateur d'accès à la carte
Indic_Interdit = 0; // Indicateur d'émission de message d'interdiction

delay(3000); // attente de 3 secondes pour permettre la lecture de
l'affichage

lcd.clear(); // efface LCD
lcd.setCursor(8, 3); // positionne curseur colonne 9, ligne 3

    if (MISE_JOUR_NTP == 1) {
        lcd.print("Avec NTP    ");
    }
    else {
        lcd.print("Sans NTP    ");
    }
}

// ----- Fin de la partie système de base du programme -----
-----

// ----- Début de la partie réception sans fil module RFLink -----
-----

// Initialisation communication RFLink
Serial2.begin(57600);
// ----- Fin de la partie réception sans fil module RFLink -----
-----

// ----- Début de la partie gestion des volets, portail et portes de garage -----
-----

// Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
// Initialisation du temps de fonctionnement des volets (limité à 20 volets)
/* EEPROM.write(0,1);
EEPROM.write(1,1);
    for (int i=2; i <= 19; i++){
        EEPROM.write(i,10);
    } */

// Initialisation de la programmation des volets
// 1000 à 1023 = prog. 1, 1024 à 1047 = prog. 2, 1048 à 1071 = prog. 3, 1072 à 1095 = prog. 4
// 1096 à 1119 = prog. 5, 1120 à 1143 = prog. 6, 1144 à 1167 = prog. 7, 1168 à 1191 = prog. 8
// 1192 à 1399 = réservé pour la possibilité de passer à 16 programmes.
/* for (i=0; i<4; i++) {
    EEPROM.write(1000 + (i * 24),0);
    EEPROM.write(1001 + (i * 24),32); // 32 pour semaine, 33 pour week-end
    EEPROM.write(1002 + (i * 24),32); // 32 pour semaine, 33 pour week-end
    EEPROM.write(1003 + (i * 24),32); // 32 pour semaine, 33 pour week-end
        for (j=0; j<16; j++) {
            EEPROM.write(1004 + (i * 24) + j,0);
        }
    EEPROM.write(1020 + (i * 24),99); // heure ouverture = 0 à 23, 99 pour aucune heure
    EEPROM.write(1021 + (i * 24),99); // minute ouverture = 0 à 59, 99 pour aucune minute
    EEPROM.write(1022 + (i * 24),99); // heure fermeture = 0 à 23, 99 pour aucune heure
    EEPROM.write(1023 + (i * 24),99); // minute fermeture = 0 à 23, 99 pour aucune minute
}
for (i=4; i<8; i++) {
    EEPROM.write(1000 + (i * 24),0);
    EEPROM.write(1001 + (i * 24),99); // 99 pour pas de date sélectionnée
    EEPROM.write(1002 + (i * 24),99);
    EEPROM.write(1003 + (i * 24),99);
        for (j=0; j<16; j++) {
            EEPROM.write(1004 + (i * 24) + j,0);
        }
    EEPROM.write(1020 + (i * 24),99); // heure ouverture = 0 à 23, 99 pour aucune heure
    EEPROM.write(1021 + (i * 24),99); // minute ouverture = 0 à 59, 99 pour aucune minute
    EEPROM.write(1022 + (i * 24),99); // heure fermeture = 0 à 23, 99 pour aucune heure
    EEPROM.write(1023 + (i * 24),99); // minute fermeture = 0 à 23, 99 pour aucune minute
} */

// Vérifie s'il ya une programmation pour les volets
charger_prog_volet(); // chargement de la programmation des volets pour la journée
// ----- Fin de la partie gestion des volets, portail et portes de garage -----
-----

// ----- Début de la partie Gestion des températures -----
-----

// Pour initialisation les paramètres de la centrale dans l'Eeprom : il faut décommenter la partie suivante (à faire lors
du premier démarrage)
// Initialisation de l'appairage des sondes de température
//    for (i=20; i <= 34; i++){
//        EEPROM.write(i,0);
//    }

// Initialisation de l'ordre des sondes de température sans fil reçues par la centrale domotique
//    for (i=35; i <= 45; i++){
//        EEPROM.write(i,0);
//    }

// Démarrage des sondes filaires
Compteur_Fil = 0; // initialisation du compteur pour la lecture des sondes filaires

    if (NB_SONDEST_F > 0) {
        dht1.begin();
    }
    if (NB_SONDEST_F > 1) {
        dht2.begin();
    }
    if (NB_SONDEST_F > 2) {
        dht3.begin();
    }
    if (NB_SONDEST_F > 3) {
        dht4.begin();
    }
    if (NB_SONDEST_F > 4) {
        dht5.begin();
    }

```

```

    }
    dht6.begin(); // sonde montée derrière la centrale domotique
    //
    // Chargement de l'ordre des sondes de température sans fil
    EnregSonde = EEPROM.read(45);
    if (EnregSonde == 1) { // Ordre enregistré
        for (i=0; i<10; i++) {
            IdT[i + 5]=EEPROM.read(35 + i); // le numéro des sondes sans fil commence à 5 (0 à 4 = sondes filaires)
        }
    }
    // initialisation du compteur pour calcul T° sondes
    Compteur_CalculT = 0;
    // Chargement du numéro des sondes appairées pour l'extérieur et l'intérieur
    NumS_Ext = EEPROM.read(20) - 1; // -1 car l'Id, laT° et l'humidité des sondes démarre à 0 en indice dans les tableaux
    NumS_Int = EEPROM.read(21) - 1; // -1 car l'Id, laT° et l'humidité des sondes démarre à 0 en indice dans les tableaux
    // ----- Fin de la partie Gestion des températures -----
}
// -----
// ***** FIN DU SETUP *****
// -----

// -----
// ***** DEBUT DU LOOP *****
// -----

void loop()
{
    // ----- Début Partie de base du programme système -----
    int i, j;

    watchdog(); // appel de la fonction reset Watchdog

    if(MISE_JOUR_NTP == 1 && Compteur_Ntp == CYCLE_NTP) { // mise à jour horloge par NTP
        calcul_ntp_time(ntp());
    }

    if (Compteur_Rtc == 0) {
        lectureRTC(); // appel de la fonction lecture données RTC DS1307
    }
    Compteur_Rtc = Compteur_Rtc + 1;
    if (Compteur_Rtc >= CYCLE_RTC) Compteur_Rtc = 0; // Reset compteur de boucle de demande auprès de l'horloge RTC DS1307

    if (JourSemaine != JourCourant) { // test si changement de jour
        JourCourant = JourSemaine; // mémorisation du jour de la semaine
        Compteur_Ntp = Compteur_Ntp + 1; // incrémentation tous les jours du compteur pour mise à jour de
l'heure par un site NTP
        // avec module DM_volets.ino
        charger_prog_volet();
    }

    TempsMn = Hours * 60 + Mins; // calcul du temps courant en minutes
    Tick_New = millis(); // Sauvegarde du tick courant

    if (Compteur_Lcd == 0){
        affiDateHeure(0); // appel de la fonction affichage date et heure dans la boucle
loop
        affichageMode(); // appel de la fonction affichage du mode de fonctionnement sur
LCD ligne 2
    }

    Compteur_Lcd = Compteur_Lcd + 1;

    if (Compteur_Lcd >= CYCLE_LCD) Compteur_Lcd = 0; //Reset compteur de boucle pour affichage.

    testLDR(); // appel de la fonction test de la luminosité => extinction de
Back Light LCD dans la pénombre

    blinkLED(TempsLED); // appel de la fonction clignotement de la LED en Face Avant

    wsServer.listen(); // Ecoute d'informations sur le port Ethernet de la WebSocket

    if (wsServer.isConnected() && (Indic_Acces == 2)) {
        Indic_Acces = 1; // L'échéance d'interdiction n'a pas été atteint
    }

    if ((wsServer.isConnected()) && (Indic_Acces == 1)) {
        // envoi des données vers le serveur web
        // Envoi des valeurs des entrées analogiques, des compteurs d'impulsions et des sondes de température, de l'alarme
vers le client WebSocket.
        if (Indic_Init_Ana == 1){
            if ((PageWeb == 1) || (PageWeb == 11)) { // page accueil et paramètres
                envoi_web_sondesT(); // envoi des sondes de température
                Indic_Init_Ana = 0; // Fin de la procédure d'envoi des informations analogiques,
des compteurs d'impulsions et des sondes de températures.
            }
            if (PageWeb == 5) { // page Chauffage
                envoi_web_sondesT(); // envoi des sondes de température
                Indic_Init_Ana = 0; // Fin de la procédure d'envoi des informations analogiques,
des compteurs d'impulsions et des sondes de températures.
            }
        }
        Compteur_Deconnexion_Nb_Periode = 0;
    }
}

```



```

else {
    if ((!wsServer.isConnected()) && (Indic_Acces == 1)) {
        // Armement d'une temporisation avant d'interdire l'accès car connexion WebSocket fermée
        Compteur_Deconnexion_Nb_Periode = NB_PERIODE_DECONNECTION;
        Indic_Acces = 2;
    }
}

// Controle de l'échéance de l'autorisation d'accès
if ((Indic_Acces == 2) && (Compteur_Deconnexion_Nb_Periode >= 0)){
    Compteur_Deconnexion_Nb_Periode --; //décrementation de l'échéance.
    if (Compteur_Deconnexion_Nb_Periode == 0){
        Indic_Acces = 0; //interdiction d'accès.
#ifdef DEBUG
        Serial.println(F("interdiction d'accès"));
#endif
    }
}

//Réponse au client en cas de demande alors qu'il y a interdiction d'accès.
if (wsServer.isConnected() && (Indic_Interdit == 1)){
    // rien pour l'instant
}

// Incrémentation et reset du compteur de cycle d'envoi des entrées analogiques,des compteurs d'impulsions et des sondes de température.
if ((Tick_New >= Tick_Ana) && ((Tick_New - Tick_Ana) >= PERIODE_CYCLE_ANA) && (Indic_Init_Ana == 0)) {
    Indic_Init_Ana = 1; //Equivaut à autoriser un nouvel envoi des entrées analogiques et des compteurs d'impulsions par le serveur.
    Tick_Ana = Tick_New;
}
else if ((Tick_New < Tick_Ana) && ((ULONG_MAX - Tick_Ana) + Tick_New >= PERIODE_CYCLE_ANA) && (Indic_Init_Ana == 0)){
    Indic_Init_Ana = 1; //Equivaut à autoriser un nouvel envoi des entrées analogiques et des compteurs d'impulsions par le serveur.
    Tick_Ana = Tick_New;
}

// ----- Fin de la partie de base du programme système -----
// ----- Début Partie réception module RFLink -----
read_reception_rflink(); // Lecture du module RFLink (433 Mhz et 2,4 Ghz)
// ----- Fin de la partie réception module RFLink -----

// ----- Début Partie Gestion des volets, portail, portes de garage -----
applique_prog_volets(); // Fonction appliquant la programmation des volets s'il y en a une
// ----- Fin de la partie Gestion des volets, portail, portes de garage -----

// ----- Début Partie gestion des températures -----
// Lecture des sondes filaires
if (Compteur_Fil == 0) { // lecture des sondes filaires à intervalle
programmé
    if(NB_SONDEST_F > 0) {
        TempT[0] = dht1.readTemperature() * 10; // lecture température en °C sonde 1
        HumT[0] = dht1.readHumidity(); // lecture humidité sonde 1
    }
    if(NB_SONDEST_F > 1) {
        TempT[1] = dht2.readTemperature() * 10; // lecture température en °C sonde 2
        HumT[1] = dht2.readHumidity(); // lecture humidité sonde 2
    }
    if(NB_SONDEST_F > 2) {
        TempT[2] = dht3.readTemperature() * 10; // lecture température en °C sonde 3
        HumT[2] = dht3.readHumidity(); // lecture humidité sonde 3
    }
    if(NB_SONDEST_F > 3) {
        TempT[3] = dht4.readTemperature() * 10; // lecture température en °C sonde 4
        HumT[3] = dht4.readHumidity(); // lecture humidité sonde 4
    }
    if(NB_SONDEST_F > 4) {
        TempT[4] = dht5.readTemperature() * 10; // lecture température en °C sonde 5
        HumT[4] = dht5.readHumidity(); // lecture humidité sonde 5
    }
    TempT[15] = dht6.readTemperature() * 10; // lecture température en °C sonde 6 (Sonde montée arrière boitier centrale)
}
Compteur_Fil = Compteur_Fil + 1;
if (Compteur_Fil >= CYCLE_FIL) Compteur_Fil = 0; // lit les condes filaires à un intervalles programmé
// fin des lecture filaire

// Traitement des températures minima et maxima.
if (Compteur_CalculT == CYCLE_CALCUL_TE) { // calcul des températures extremes à intervalle
programmé
    if (int(TempT[NumS_Int] != 0)) {
        if (int(TempT[NumS_Int]) > Temp_Int_Maxi) Temp_Int_Maxi = int(TempT[NumS_Int]);
        if (int(TempT[NumS_Int]) < Temp_Int_Mini) Temp_Int_Mini = int(TempT[NumS_Int]);
    }
    if (int(TempT[NumS_Ext] != 0)) {
        if (int(TempT[NumS_Ext]) > Temp_Ext_Maxi) Temp_Ext_Maxi = int(TempT[NumS_Ext]);
        if (int(TempT[NumS_Ext]) < Temp_Ext_Mini) Temp_Ext_Mini = int(TempT[NumS_Ext]);
    }
}
//
Compteur_CalculT = Compteur_CalculT + 1;
if (Compteur_CalculT > CYCLE_CALCUL_TE) Compteur_CalculT = 0; // calcule les extremes des température à un intervalles programmé
// ----- Fin Partie gestion des températures -----

```

```

// ----- Partie servant au débogage -----
-----
#ifdef DEBUG_MEM
    Compteur_Test ++;
    if (Compteur_Test >= 1000){
        Compteur_Test = 0;
        Serial.print(F("Memoire Ram dispo="));
        Serial.println(freeRam());
    }
#endif // DEBUG
}
// -----
// ***** FIN DU LOOP *****
// -----

// -----
// ***** DEBUT DES FONCTIONS *****
// -----

// ***** Début des fonctions système, indispensables au fonctionnement du programme *****

// ----- début fonctions associées au WebSocket -----
-----
void onConnect(WebSocket &socket)
{
    #ifdef DEBUG
        Serial.println(F("onConnect called"));          // pour test
    #endif
}

// Programme de traitement des données reçues du client websocket
// ----- début de la fonction onData (Traitement des messages reçus -----
-----
void onData(WebSocket &socket, char* DataString, byte FrameLength)
{
    byte Rang = 0;
    char Buff[3];

    #ifdef DEBUG
        Serial.print(F("chaine recue : "));
        Serial.write((unsigned char*)DataString, FrameLength);
        Serial.println();
    #endif

    if ((DataString[0] == (STX)) && (DataString[1] == NumCard[0]) && (DataString[2] == NumCard[1]) && (DataString[3] ==
NumCard[2]) && (DataString[4] == NumCard[3])) {
        // test si code d'accès
        if (strncmp(&DataString[6], "PAGEC", 5) == 0){
            Buff[0] = DataString[12];
            Buff[1] = DataString[13];
            Buff[2] = '\0';
            PageWeb = atoi(Buff);
            #ifdef DEBUG
                Serial.print(F("Page : "));
                Serial.println(PageWeb);
            #endif
        }
        if (strncmp(&DataString[6], "ACCES", 5) == 0){
            #ifdef DEBUG
                Serial.print(F("demande d'accès="));
                Serial.println(DataString);
            #endif
            // test si le code d'accès est valide
            if (strncmp(&DataString[12], CODE_CARTE, 4) == 0){
                Indic_Acces = 1;
                Compteur_Deconnexion_Nb_Periode = 0;
                // Elaboration du message d'autorisation
                EvtServer[0] = STX;
                strncpy(&EvtServer[1], NumCard, 4);
                EvtServer[5] = Separator;
                strncpy(&EvtServer[6], "REACC", 5);
                EvtServer[11] = Separator;
                strncpy(&EvtServer[12], "AUT", 3);
                EvtServer[15] = ETX;
                EvtServer[16] = '\0';
                #ifdef DEBUG
                    Serial.println(F("Code OK"));
                    Serial.print(F("Chaine emise :"));
                    Serial.println(EvtServer);
                #endif
            }
            else {
                Indic_Interdit = 1;
                // Elaboration du message d'interdiction
                EvtServer[0] = STX;
                strncpy(&EvtServer[1], NumCard, 4);
                EvtServer[5] = Separator;
                strncpy(&EvtServer[6], "REACC", 5);
                EvtServer[11] = Separator;
                strncpy(&EvtServer[12], "INT", 3);
                EvtServer[15] = ETX;
                EvtServer[16] = '\0';
                #ifdef DEBUG
                    Serial.println(F("Code faux"));
                    Serial.print(F("Chaine emise :"));
                    Serial.println(EvtServer);
                #endif
            }
        }
    }
}

```

```

        #endif
    }
    // Vérification de la connexion du client
    if (socket.isConnected()) {
        socket.send(EvtServer,16);
    }
}
if (Indic_Acces == 0) {
    Indic_Interdit = 1; // pour envoi d'une réponse d'interdiction d'accès
}
// Accès autorisé, donc applique les ordres reçus
if (Indic_Acces==1) {
    if (strcmp(&DataString[6],"PORTA",5) == 0) { // Réception d'une demande de commande du portail
        commandePortail(); // Commande du portail
    }
    else if (strcmp(&DataString[6],"GARAG",5) == 0) { //Réception d'une demande de commande de porte de
garage
        Buff[0] = DataString[12];
        Buff[1] = DataString[13];
        Buff[2] = '\0';
        Rang = atoi(Buff);
        commandeGarage(Rang); // Commande de la porte de garage demandée
    }
    else if (strcmp(&DataString[6],"VOLET",5) == 0) { // Réception d'une demande de commande des volets
        gestion_volets(DataString); // appel de la fonction de la gestion des volets
    }
    else if (strcmp(&DataString[6],"DPROV",5) == 0) { // Demande d'envoi des données de programmation
volets à l'interface web
        envoiProgVolet();
    }
    else if (strcmp(&DataString[6],"ENPRV",5) == 0) { // appel de la procédure de mise en mémoire des
programmations volets envoyés par l'interface web
        enregProgrammeVolet(DataString);
    }
    else if (strcmp(&DataString[6],"DPARA",5) == 0) { // Réception d'une demande d'envoi des paramètres
de l'unité de gestion à l'interface web
        #ifdef DEBUG
            Serial.println(F("Demandes des parametres"));
        #endif
        Buff[0] = DataString[12];
        Buff[1] = DataString[13];
        Buff[2] = '\0';
        PageWeb = atoi(Buff);
        if (PageWeb == 1) { // Page Index
            envoi_appairage_sondesT(5); // Envoi de l'appairage des sondes T° des 5
premières pièces
        }
        if (PageWeb == 5) { // page Chauffage
            // Demande d'envoi de l'appairage des sondes de température (par rapport au nombre de pièces de
l'habitation)
            envoi_appairage_sondesT(12);
        }
        if (PageWeb == 11) { // Page Paramètres
            envoiParametresVolets(); // Envoi des paramètres d'initialisation des
volets
            envoi_appairage_sondesT(15); // Envoi de l'appairage des 15 sondes T°
        }
    }
    else if (strcmp(&DataString[6],"EPARA",5) == 0) { // appel de la procédure de mise en mémoire des
paramètres envoyés par l'interface web
        Buff[0] = DataString[12];
        Buff[1] = DataString[13];
        Buff[2] = '\0';
        PageWeb = atoi(Buff);
        if (PageWeb == 1) {
            sauveParametresVolets(DataString); // Enregistrement dans l'Eeprom de
l'initialisation des volets
        }
        if (PageWeb == 2) {
            sauve_appairage_sondesT(DataString); // Enregistrement dans l'Eeprom de l'appairage des
sondes de température
        }
    }
    else if (strcmp(&DataString[6],"INITI",5) == 0) { // appel de la procédure de réinitialisation des
sondes de température
        Buff[0] = DataString[12];
        Buff[1] = DataString[13];
        Buff[2] = '\0';
        Rang = atoi(Buff);
        if (Rang == 1) { // 1 pour rinitialisation des sondes de température
            initSondesT();
        }
    }
    else if (strcmp(&DataString[6],"CMDPR",5) == 0) { // Commande des prises électriques
        commande_prises(DataString);
    }
}
}
}
// ----- fin de la fonction onData -----

// ----- Début de la fonction onDisconnect (Déconnexion du webSocket) -----
void onDisconnect(Websocket &socket)
{
    #ifdef DEBUG
        Serial.println(F("onDisconnect called"));
    #endif
}
// ----- fin de la fonction onDisconnect -----

```



```

// ----- Fin fonctions associées au WebSocket -----
// -----
// ----- Début de la fonction ntp , retourne le nombre de sec. depuis 1 Jan 1970. (Unix time) -----
// -----
unsigned long ntp()
{
    const int NTP_PACKET_SIZE= 48; // NTP time stamp is in the first 48 bytes of the message
    byte PacketBuffer[NTP_PACKET_SIZE]; //buffer to hold incoming and outgoing packets

    EthernetUDP Udp;
    memset(PacketBuffer, 0, NTP_PACKET_SIZE); //initialise toute les "case" du tableau à 0

    //construction de la requete
    PacketBuffer[0] = 0b11100011; // LI, Version, Mode
    PacketBuffer[1] = 0; // Stratum, or type of clock
    PacketBuffer[2] = 6; // Polling Interval
    PacketBuffer[3] = 0xEC; // Peer Clock Precision
    // 8 bytes of zero for Root Delay & Root Dispersion
    PacketBuffer[12] = 49;
    PacketBuffer[13] = 0x4E;
    PacketBuffer[14] = 49;
    PacketBuffer[15] = 52;

    Udp.begin(PORT_TIME_SERVER);
    Udp.beginPacket(TIME_SERVER, 123); // NTP requests are to port 123
    Udp.write(PacketBuffer,NTP_PACKET_SIZE);
    Udp.endPacket();

    delay(1000);
    if ( Udp.parsePacket() )
    {
        Udp.read(PacketBuffer,NTP_PACKET_SIZE);
        Udp.stop();
        const unsigned long seventyYears = 2208988800UL;
        unsigned long highWord = word(PacketBuffer[40], PacketBuffer[41]);
        unsigned long lowWord = word(PacketBuffer[42], PacketBuffer[43]);
        unsigned long secsSince1900 = highWord << 16 | lowWord;
        return secsSince1900 - seventyYears;
    }
    else
    {
        lcd.clear(); // efface LCD
        lcd.setCursor(0,0); // positionne curseur colonne 1, ligne 1
        lcd.print("Pb NTP");
    }
    return 0;
}
// ----- fin de la fonction ntp -----
// -----
// ----- début de la fonction fonction recherche NTP et mise à jour RTC -----
// -----
void calcul_ntp_time(unsigned long Epoch)
{
    lcd.setCursor(8,3); // positionne curseur colonne 10, ligne 4
    if (Epoch == 0)
    {
        lcd.print("Defaut NTP ");
    }
    else
    {
        calcul_DLS();
        // Remets à l'heure le RTC DS1307
        RTC.set(Epoch + 3600 + (Dls * 3600));
        Compteur_Ntp = 0; // remise à jour du compteur NTP
    }
}
// ----- fin de la fonction fonction recherche NTP et mise à jour RTC -----
// -----
// ----- début de la fonction calcul DLS -----
// -----
// fonction qui calcule DLS (Day Light Saving : heure d'hiver DLS = 0 , d'heure d'été DLS = 1)void calculDLS()
void calcul_DLS()
{
    if ((Mois > 3) && (Mois < 10) || ((Mois == 3) && (DateJour > 24) && (((DateJour - JourSemaine) > 24) || (JourSemaine == 7))) || ((Mois == 10) && ((DateJour < 25) || (((DateJour - JourSemaine) < 25) && (JourSemaine != 7))))
    {
        Dls = 1;
    }
    else
    {
        Dls = 0;
    }
}
// ----- Fin de la fonction calcul DLS -----
// -----
// ----- début de la fonction lecture_rtc -----
// -----
// Fonction de lecture du module Horloge DS1307RTC avec mise à jour des variables Temps
void lectureRTC()
{
    Epoch_Local = RTC.get();
    Hours = hour(Epoch_Local);
    Mins = minute(Epoch_Local);
    Sec = second(Epoch_Local);
    DateJour = day(Epoch_Local);
    Mois = month(Epoch_Local);
    Annee = year(Epoch_Local);
    JourSemaine = weekday(Epoch_Local);
}

```

```

}
// ----- fin de la fonction lecture RTC -----
-----

// -----début de la fonction affiDateHeure (affichage de la date et de l'heure dans le LCD) -----
-----
void affiDateHeure(int Ligne)
{
    int AnneeCours;

    lcd.setCursor(0,Ligne);                // positionne le curseur colonne 1, ligne demandée
    // affichage jour de la semaine
    switch (JourSemaine)
    {
        case 1:
            lcd.print("Dim. ");
            break;
        case 2:
            lcd.print("Lun. ");
            break;
        case 3:
            lcd.print("Mar. ");
            break;
        case 4:
            lcd.print("Mer. ");
            break;
        case 5:
            lcd.print("Jeu. ");
            break;
        case 6:
            lcd.print("Ven. ");
            break;
        case 7:
            lcd.print("Sam. ");
            break;
    }
    // affichage jour
    if (DateJour < 10)
    {
        lcd.print('0');
    }
    lcd.print(DateJour);
    lcd.print('/');
    // affichage mois
    if (Mois < 10)
    {
        lcd.print('0');
    }
    lcd.print(Mois);
    lcd.print('/');
    // affichage année
    AnneeCours = Annee - 2000;
    lcd.print(AnneeCours);
    //
    lcd.print(' ');
    // affichage heures
    if (Hours < 10)
    {
        lcd.print('0');
    }
    lcd.print(Hours);
    lcd.print(':');
    // affichage minutes
    if (Mins < 10)
    {
        lcd.print('0');
    }
    lcd.print(Mins);
}
// ----- fin de la fonction affiDateHeure -----
-----

// ----- début de la fonction affichage du mode de fonctionnement LCD ligne 2 -----
-----
static inline void affichageMode()
{
    lcd.setCursor(0, 1);                // positionne curseur colonne 1, ligne 2
    if (TempsLED == TEMPS_LED_NORMAL) {
        lcd.print("Mode Normal  ");
    }
    else {
        lcd.print("Mode Defaut  ");
    }
    // affichage de la température du boitier
    lcd.print(TempT[15]/10);
}
// ----- fin de la fonction affichage du mode de fonctionnement LCD ligne 2 -----
-----

// ----- début de la fonction testLDR (test de la luminosité) -----
-----
static inline void testLDR()
{
    if (analogRead(PIN_LDR) > 500) {
        if (TempsMn > TempsLDR) {
            digitalWrite (PIN_BLLCD,0);    // Back Light LCD éteint
        }
    }
    else{
        digitalWrite (PIN_BLLCD,1);        // Back Light LCD allumé
        TempsLDR = TempsMn ;
    }
}
}

```

```

// ----- fin de la fonction testLDR -----
// -----
// ----- début de la fonction blinkLED (fait clignoter la LED) -----
// -----

void blinkLED(int TempsClign)
{
    static unsigned long PrevMillis = 0;          // déclaration d'une variable pour la temporisation de la LED
    static boolean EtatLed = true;                // déclaration d'une variable pour la temporisation de la LED

    unsigned long CurMillis = millis();
    if(CurMillis - PrevMillis > TempsClign) {
        PrevMillis = CurMillis;
        if(EtatLed) {
            digitalWrite (PIN_LED_WATCHDOG,LOW);
            EtatLed = false;
        }
        else {
            digitalWrite (PIN_LED_WATCHDOG,HIGH);
            EtatLed = true;
        }
    }
}
// ----- fin de la fonction blinkLED -----
// -----

// ----- début de la fonction test carte SD -----
// -----
void testCarteSD()
{
    lcd.setCursor(0,3);                          // positionne curseur colonne 1, ligne 4
    TestSD = SD.begin(PIN_SELECT_SD);             // initialisation de la carte SD avec broche 4 en tant que CS - renvoie true/false
    if (TestSD != true) {                         // si initialisation n'est pas réussie
        lcd.print("SD Pb");
        CarteSD = false;
        TempsLED = TEMPS_LED_PANNE;              // fonctionnement anomalie
    }
    else {                                        // si initialisation réussie
        lcd.print("SD OK");
        CarteSD = true;
    } // fin si SD.begin
}
// ----- fin de la fonction test catre SD -----
// -----

// ----- début de la fonction Reset Watchdog -----
// -----
void watchdog()
{
    digitalWrite (PIN_WATCHDOG, HIGH);
    delay (2);
    digitalWrite (PIN_WATCHDOG, LOW);             // Reset Watchdog
}
// ----- fin de la fonction Reset Watchdog -----
// -----

// ----- Début de la fonction freeRam : Calcul mémoire ram dynamique -----
// -----
int freeRam ()
{
    extern int __heap_start, *__brkval;
    int v;
    return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
// ----- fin de la fonction freeRam -----
// -----
// -----
// ***** FIN DES FONCTIONS DU PROGRAMME SYSTEME *****
// -----
// -----
// -----
// ***** DEBUT DES FONCTIONS COMMUNES A PLUSIEURS MODULES *****
// -----
// -----
void creMessage(char* NumRang, char* Message, byte LongMes)
{
    EvtServer[0] = STX;
    strncpy(&EvtServer[1], NumCard, 4);
    EvtServer[5] = Separator;
    strncpy(&EvtServer[6], "MESSA",5);
    EvtServer[11] = Separator;
    strncpy(&EvtServer[12], NumRang, 2);
    EvtServer[14] = Separator;
    strncpy(&EvtServer[15], Message, LongMes);
    EvtServer[15 + LongMes] = ETX;
    EvtServer[16 + LongMes] = '\0';
    wsServer.send(EvtServer,16 + LongMes);
    #ifdef DEBUG
        Serial.print(F("chaine emise = "));
        Serial.println(EvtServer);
    #endif
}
// ----- fin de la fonction creMessage -----
// -----

// ----- Début de la fonction read_reception_relink (lecture de la réception des données du module RFLink -----
// -----

```

```

void read_reception_rflink()
{
    char DebSignal = 0x0a;          // caractère LF en hexadécimal
    char FinSignal = 0x0d;          // caractère CR en hexadécimal
    byte i;

    if (Serial2.available() > 0) {
        InByte2 = Serial2.read();
        if (InByte2 == DebSignal) {
            Bufflen2 = 0;
        }
        else {
            Buffin2[Bufflen2] = InByte2;
            Bufflen2++;
        }
        if (Bufflen2 > 80) {
            Bufflen2 = 0;          // traitement erreur
        }
        else if (InByte2 == FinSignal && Bufflen2 > 5) {      // fin de ligne trouvée
            Buffin2[Bufflen2] = '\0';          // caractère de fin de chaîne
            #ifdef DEBUG
                Serial.print(F("RFLink = "));
                Serial.write(Buffin2);
                Serial.println();
            #endif
            traiteRFLink(Buffin2);
        }
    }
}

// ----- Fin de la fonction read_reception_rflink (lecture de la réception des données du module RFLink) -----
// ----- Début de la fonction traiteRFLink (traitement des données reçues du module RFLink) -----

void traiteRFLink(char* ChRecue)
{
    // Interprétation de la chaîne reçue
    if (strcmp(&ChRecue[0], "20;", 3) == 0) {                  // Réception d'une donnée valide RFLink
        if ((strcmp(&ChRecue[6], "Oregon Temp;", 12) == 0) || (strcmp(&ChRecue[6], "Oregon TempHygro", 16) == 0)) {
            // Réception d'une donnée valide RFLink
            traiteSondesTemperature(ChRecue);                 // dans module DM_temperature
        }
        else if ((strcmp(&ChRecue[6], "Chuango", 7) == 0) || (strcmp(&ChRecue[6], "Xcellent", 8) == 0)) {      //
            // marque des différents capteurs alarme
            //traiteAlarme(ChRecue);                            // dans module DM_protection
        }
        else if (strcmp(&ChRecue[0], "50;", 3) == 0) {         // Réception d'une donnée du clavier déporté
            if (strcmp(&ChRecue[3], "NRFcl;", 6) == 0) {        // Identification du clavier
                if (strcmp(&ChRecue[9], "ETAT", 4) == 0) {      // Demande d'envoi de l'état de l'alarme
                    //envoi_etatAlarme();
                }
                else if (strcmp(&ChRecue[9], "METAT", 5) == 0) {
                    //activeAlarme(ChRecue, 'C');               // C pour demande faite par le clavier déporté
                }
            }
        }
    }
}

// ----- Fin de la fonction traiteRFLink (traitement des données reçues du module RFLink) -----
// -----
// ***** FIN DES FONCTIONS COMMUNES A PLUSIEURS MODULES *****
// -----

```