Projet_VAE_Nice\terminal_patient.cpp

```cpp
#include "terminal_patient.h"

TerminalPatient::TerminalPatient(uint16_t id, uint8_t chambre)
  : terminalId(id), chambreId(chambre) {}

void TerminalPatient::begin() {
  // Init Serial
  Serial.begin(115200);

  // Init LoRa
  SPI.begin(5, 19, 27, 18);  // SCK, MISO, MOSI, SS
  LoRa.setPins(18, 14, 26);  // SS, RST, DIO0

  if (!LoRa.begin(LORA_FREQUENCY)) {
    Serial.println("Erreur init LoRa!");
    while (1);
  }

  LoRa.setSyncWord(LORA_SYNC_WORD);
  LoRa.setTxPower(LORA_TX_POWER);
  LoRa.setSpreadingFactor(LORA_SPREADING_FACTOR);

  // Init TFT
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(TFT_BLACK);

  // Init préférences (EEPROM)
  preferences.begin("terminal", false);

  // Config WiFi (backup)
  WiFi.mode(WIFI_STA);

  // Init GPIO
  pinMode(33, INPUT_PULLUP);  // Bouton urgence
  pinMode(25, OUTPUT);        // LED RGB

  // Afficher menu principal
  drawMainMenu();

  Serial.println("Terminal patient initialisé");
}

void TerminalPatient::loop() {
  // 1. Vérifier bouton urgence
  handleEmergencyButton();

  // 2. Vérifier écran tactile
  handleTouch();

  // 3. Écouter accusés de réception LoRa
```

```
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    String ack = "";
    while (LoRa.available()) {
      ack += (char)LoRa.read();
    }
    if (ack == "ACK") {
      updateStatusLED(true);
    }
  }

  delay(50);  // Éviter la surcharge CPU
}

bool TerminalPatient::sendLoRaMessage(Besoin besoin, uint8_t priorite) {
  LoraMessage msg;
  msg.terminalId = terminalId;
  msg.chambreId = chambreId;
  msg.besoin = besoin;
  msg.priorite = priorite;
  msg.timestamp = millis();

  // Calcul checksum
  uint8_t* data = (uint8_t*)&msg;
  msg.checksum = calculateChecksum(data, sizeof(msg) - sizeof(uint16_t));

  // Envoi LoRa
  LoRa.beginPacket();
  LoRa.write((uint8_t*)&msg, sizeof(msg));
  bool success = LoRa.endPacket() > 0;

  if (success) {
    Serial.println("Message LoRa envoyé");

    // Backup HTTP si WiFi disponible
    if (isNetworkAvailable()) {
      sendHTTPBackup(String("Chambre ") + chambreId + " - Besoin: " + String(besoin));
    }

    logEvent("Message envoyé - Besoin: " + String(besoin));
    showConfirmation(besoin);
  }

  return success;
}

void TerminalPatient::handleTouch() {
  uint16_t x, y;

  // Lecture écran tactile résistif
  // (Simplifié - besoin d'un ADC pour lecture précise)
  if (digitalRead(13) == LOW) {  // Détection touch
    // Calculer coordonnées
    x = analogRead(14);
```

```
    y = analogRead(27);

    // Déterminer bouton pressé
    if (x > 1000 && x < 2000) {
      if (y > 1000 && y < 1500) {
        sendLoRaMessage(BESOIN_DOULEUR, 2);
      } else if (y > 1500 && y < 2000) {
        sendLoRaMessage(BESOIN_TOILETTE, 1);
      }
      // ... autres zones
    }

    delay(300);  // Anti-rebond
  }
}

void TerminalPatient::handleEmergencyButton() {
  if (digitalRead(33) == LOW &&
      (millis() - lastEmergencyPress) > DEBOUNCE_DELAY) {
    lastEmergencyPress = millis();

    // Envoyer urgence avec priorité max
    if (sendLoRaMessage(BESOIN_URGENCE, 3)) {
      // Activer buzzer si présent
      tone(26, 2000, 1000);
    }
  }
}

void TerminalPatient::drawMainMenu() {
  tft.fillScreen(TFT_BLACK);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setTextSize(2);

  // En-tête
  tft.drawCentreString("BESOIN ?", 120, 10, 2);
  tft.drawLine(0, 40, 240, 40, TFT_WHITE);

  // Boutons
  tft.fillRoundRect(20, 50, 100, 50, 10, TFT_RED);
  tft.drawCentreString("DOULEUR", 70, 65, 2);

  tft.fillRoundRect(140, 50, 100, 50, 10, TFT_BLUE);
  tft.drawCentreString("TOILETTE", 190, 65, 2);

  tft.fillRoundRect(20, 120, 100, 50, 10, TFT_GREEN);
  tft.drawCentreString("EAU", 70, 135, 2);

  tft.fillRoundRect(140, 120, 100, 50, 10, TFT_ORANGE);
  tft.drawCentreString("AIDE", 190, 135, 2);

  // Pied de page
  tft.drawLine(0, 190, 240, 190, TFT_WHITE);
  tft.drawCentreString("Chambre " + String(chambreId), 120, 200, 2);
```

```
}

void TerminalPatient::showConfirmation(Besoin besoin) {
  tft.fillScreen(TFT_BLACK);
  tft.setTextColor(TFT_GREEN, TFT_BLACK);
  tft.setTextSize(3);
  tft.drawCentreString("MESSAGE", 120, 50, 3);
  tft.drawCentreString("ENVOYE", 120, 90, 3);

  // Afficher le besoin
  tft.setTextSize(2);
  String besoinStr = "";
  switch(besoin) {
    case BESOIN_DOULEUR: besoinStr = "Douleur"; break;
    case BESOIN_TOILETTE: besoinStr = "Toilette"; break;
    case BESOIN_URGENCE: besoinStr = "URGENCE"; break;
    default: besoinStr = "Besoin";
  }
  tft.drawCentreString(besoinStr, 120, 140, 2);

  delay(2000);
  drawMainMenu();
}

uint16_t TerminalPatient::calculateChecksum(const uint8_t* data, size_t len) {
  uint16_t crc = 0xFFFF;
  for (size_t i = 0; i < len; i++) {
    crc ^= data[i];
    for (int j = 0; j < 8; j++) {
      if (crc & 0x0001) {
        crc = (crc >> 1) ^ 0xA001;
      } else {
        crc = crc >> 1;
      }
    }
  }
  return crc;
}

void TerminalPatient::updateStatusLED(bool confirmed) {
  // LED RGB - WS2812B
  if (confirmed) {
    // Vert = message reçu
    neopixelWrite(25, 0, 20, 0);
  } else {
    // Rouge clignotant = en attente
    static bool ledState = false;
    ledState = !ledState;
    if (ledState) {
      neopixelWrite(25, 20, 0, 0);
    } else {
      neopixelWrite(25, 0, 0, 0);
    }
  }
```

```
}

void TerminalPatient::logEvent(const String& event) {
  preferences.putString("last_event", event);
  preferences.putUInt("event_count",
  preferences.getUInt("event_count", 0) + 1);
}
•   main.ino

#include "terminal_patient.h"

// Configuration unique par terminal
#define TERMINAL_ID 1001  // Unique ID
#define CHAMBRE_NUM 12    // Numéro de chambre

TerminalPatient terminal(TERMINAL_ID, CHAMBRE_NUM);

void setup() {
  terminal.begin();
}

void loop() {
  terminal.loop();
}

```