Projet_VAE_Nice\gateway_lora.cpp

```cpp
#include "gateway_lora.h"
#include "terminal_patient.h"  // Pour LoraMessage

LoRaGateway::LoRaGateway(const char* ssid, const char* password,
                         const char* serverUrl)
  : wifiSSID(ssid), wifiPassword(password), apiServerUrl(serverUrl) {}

void LoRaGateway::begin() {
  Serial.begin(115200);

  // Init LoRa
  SPI.begin(5, 19, 27, 18);
  LoRa.setPins(18, 14, 26);

  if (!LoRa.begin(LORA_FREQUENCY)) {
    Serial.println("Gateway: Erreur LoRa!");
    return;
  }

  LoRa.setSyncWord(LORA_SYNC_WORD);
  LoRa.onReceive([](int packetSize) {
    // Callback - traité dans loop()
  });

  LoRa.receive();

  // Connexion WiFi
  connectWiFi();

  Serial.println("Gateway LoRa initialisée");
}

void LoRaGateway::loop() {
  // Traitement LoRa
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    processLoRaPacket();
  }

  // Envoi queue vers serveur
  if (!messageQueue.empty() && WiFi.status() == WL_CONNECTED) {
    for (auto it = messageQueue.begin(); it != messageQueue.end(); ) {
      if (sendToServer(*it)) {
        messagesForwarded++;
        it = messageQueue.erase(it);
      } else {
        ++it;
      }
    }
  }

```

```
  // Affichage statut périodique
  if (millis() - lastStatusUpdate > 5000) {
    displayStatus();
    lastStatusUpdate = millis();
  }
}

void LoRaGateway::processLoRaPacket() {
  ReceivedMessage recvMsg;

  // Lire données LoRa
  if (LoRa.available() >= sizeof(LoraMessage)) {
    LoRa.readBytes((uint8_t*)&recvMsg.loraData, sizeof(LoraMessage));

    // Vérifier checksum
    uint16_t receivedChecksum = recvMsg.loraData.checksum;
    recvMsg.loraData.checksum = 0;

    uint16_t calculatedChecksum = calculateChecksum(
      (uint8_t*)&recvMsg.loraData, sizeof(LoraMessage) - sizeof(uint16_t));

    if (receivedChecksum == calculatedChecksum) {
      // Message valide
      recvMsg.rssi = LoRa.packetRssi();
      recvMsg.snr = LoRa.packetSnr();
      recvMsg.receiveTime = millis();

      messagesReceived++;

      // Ajouter à queue
      if (messageQueue.size() < MAX_QUEUE_SIZE) {
        messageQueue.push_back(recvMsg);
      }

      // Envoyer accusé LoRa
      sendLoRaAck(recvMsg.loraData.terminalId);

      Serial.printf("Message reçu: Terminal %d, Chambre %d, Besoin %d\n",
                    recvMsg.loraData.terminalId,
                    recvMsg.loraData.chambreId,
                    recvMsg.loraData.besoin);
    }
  }
}

bool LoRaGateway::sendToServer(const ReceivedMessage& msg) {
  HTTPClient http;

  // Préparer JSON
  JsonDocument doc;
  doc["terminal_id"] = msg.loraData.terminalId;
  doc["chambre"] = msg.loraData.chambreId;
  doc["besoin"] = msg.loraData.besoin;
  doc["priorite"] = msg.loraData.priorite;
```

```
  doc["timestamp"] = msg.loraData.timestamp;
  doc["rssi"] = msg.rssi;
  doc["snr"] = msg.snr;

  String jsonStr;
  serializeJson(doc, jsonStr);

  // Envoyer HTTP POST
  http.begin(wifiClient, String(apiServerUrl) + "/api/alerte");
  http.addHeader("Content-Type", "application/json");

  int httpCode = http.POST(jsonStr);

  bool success = (httpCode == 200);
  if (success) {
    Serial.println("Message forwardé au serveur");
  } else {
    Serial.printf("Erreur HTTP: %d\n", httpCode);
  }

  http.end();
  return success;
}

void LoRaGateway::sendLoRaAck(uint16_t terminalId) {
  LoRa.beginPacket();
  LoRa.print("ACK:");
  LoRa.print(terminalId);
  LoRa.endPacket();
}

void LoRaGateway::connectWiFi() {
  Serial.printf("Connexion au WiFi: %s\n", wifiSSID);

  WiFi.begin(wifiSSID, wifiPassword);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 20) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.printf("\nConnecté! IP: %s\n", WiFi.localIP().toString().c_str());
  } else {
    Serial.println("\nÉchec connexion WiFi");
  }
}

void LoRaGateway::displayStatus() {
  Serial.println("\n=== STATUT GATEWAY ===");
  Serial.printf("WiFi: %s\n", WiFi.status() == WL_CONNECTED ? "OK" : "KO");
  Serial.printf("Messages reçus: %lu\n", messagesReceived);
```

```
160   Serial.printf("Messages envoyés: %lu\n", messagesForwarded);
161   Serial.printf("Queue: %d messages\n", messageQueue.size());
162   Serial.println("====================\n");
163 }
164
```