Projet_VAE_Nice\serveur_api.cpp

```cpp
// Plateforme Serveur API NurseCall avec Crow et SQLite
#include <crow.h>
#include <sqlite3.h>
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>
#include <chrono>
#include <ctime>

class NurseCallServer {
private:
    sqlite3* db;
    std::mutex db_mutex;

public:
    NurseCallServer() {
        // Initialiser base de données
        int rc = sqlite3_open("nursecall.db", &db);
        if (rc) {
            std::cerr << "Erreur DB: " << sqlite3_errmsg(db) << std::endl;
            return;
        }

        // Créer tables
        const char* sql = R"(
            CREATE TABLE IF NOT EXISTS alertes (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                terminal_id INTEGER,
                chambre INTEGER,
                besoin INTEGER,
                priorite INTEGER,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                traite BOOLEAN DEFAULT 0,
                soignant TEXT
            );

            CREATE TABLE IF NOT EXISTS terminaux (
                id INTEGER PRIMARY KEY,
                chambre INTEGER UNIQUE,
                derniere_activite DATETIME,
                statut TEXT DEFAULT 'actif'
            );

            CREATE TABLE IF NOT EXISTS logs (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                niveau TEXT,
                message TEXT,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
            );
        )";
```

```
52
53          char* errMsg = 0;
54          rc = sqlite3_exec(db, sql, 0, 0, &errMsg);
55          if (rc != SQLITE_OK) {
56              std::cerr << "Erreur SQL: " << errMsg << std::endl;
57              sqlite3_free(errMsg);
58          }
59      }
60
61      ~NurseCallServer() {
62          sqlite3_close(db);
63      }
64
65      void log(const std::string& level, const std::string& message) {
66          std::lock_guard<std::mutex> lock(db_mutex);
67
68          const char* sql = "INSERT INTO logs (niveau, message) VALUES (?, ?)";
69          sqlite3_stmt* stmt;
70
71          if (sqlite3_prepare_v2(db, sql, -1, &stmt, 0) == SQLITE_OK) {
72              sqlite3_bind_text(stmt, 1, level.c_str(), -1, SQLITE_STATIC);
73              sqlite3_bind_text(stmt, 2, message.c_str(), -1, SQLITE_STATIC);
74              sqlite3_step(stmt);
75              sqlite3_finalize(stmt);
76          }
77      }
78
79      void run(int port = 5000) {
80          crow::SimpleApp app;
81
82          // Route pour recevoir alertes ESP32
83          CROW_ROUTE(app, "/api/alerte").methods("POST"_method)
84          ([this](const crow::request& req) {
85              auto x = crow::json::load(req.body);
86              if (!x) {
87                  return crow::response(400, "JSON invalide");
88              }
89
90              int terminal_id = x["terminal_id"].i();
91              int chambre = x["chambre"].i();
92              int besoin = x["besoin"].i();
93              int priorite = x["priorite"].i();
94
95              // Insérer en base
96              std::lock_guard<std::mutex> lock(db_mutex);
97              const char* sql = R"(
98                  INSERT INTO alertes
99                  (terminal_id, chambre, besoin, priorite)
100                 VALUES (?, ?, ?, ?)
101             )";
102
103             sqlite3_stmt* stmt;
104             if (sqlite3_prepare_v2(db, sql, -1, &stmt, 0) == SQLITE_OK) {
105                 sqlite3_bind_int(stmt, 1, terminal_id);
```

```
                sqlite3_bind_int(stmt, 2, chambre);
                sqlite3_bind_int(stmt, 3, besoin);
                sqlite3_bind_int(stmt, 4, priorite);
                sqlite3_step(stmt);
                sqlite3_finalize(stmt);

                log("INFO", "Alerte reçue chambre " + std::to_string(chambre));

                // WebSocket broadcast aux clients
                broadcastWebSocket(chambre, besoin, priorite);

                return crow::response(200, "{\"status\":\"ok\"}");
            }

            return crow::response(500, "Erreur DB");
        });

        // Route pour dashboard
        CROW_ROUTE(app, "/api/alertes/en-cours")
        ([this]() {
            std::lock_guard<std::mutex> lock(db_mutex);
            crow::json::wvalue result;

            const char* sql = R"(
                SELECT a.*, t.statut
                FROM alertes a
                LEFT JOIN terminaux t ON a.terminal_id = t.id
                WHERE a.traite = 0
                ORDER BY a.priorite DESC, a.timestamp ASC
                LIMIT 50
            )";

            sqlite3_stmt* stmt;
            if (sqlite3_prepare_v2(db, sql, -1, &stmt, 0) == SQLITE_OK) {
                int idx = 0;
                while (sqlite3_step(stmt) == SQLITE_ROW) {
                    crow::json::wvalue row;
                    row["id"] = sqlite3_column_int(stmt, 0);
                    row["chambre"] = sqlite3_column_int(stmt, 2);
                    row["besoin"] = sqlite3_column_int(stmt, 3);
                    row["priorite"] = sqlite3_column_int(stmt, 4);
                    row["timestamp"] = reinterpret_cast<const char*>(
                        sqlite3_column_text(stmt, 5));

                    result[idx++] = row;
                }
                sqlite3_finalize(stmt);
            }

            return result;
        });

        // Route pour prise en charge
        CROW_ROUTE(app, "/api/alerte/<int>/prendre").methods("POST"_method)
```

```
        ([this](int alerte_id, const crow::request& req) {
            auto x = crow::json::load(req.body);
            if (!x || !x.has("soignant")) {
                return crow::response(400);
            }

            std::string soignant = x["soignant"].s();

            std::lock_guard<std::mutex> lock(db_mutex);
            const char* sql = "UPDATE alertes SET traite=1, soignant=? WHERE id=?";

            sqlite3_stmt* stmt;
            if (sqlite3_prepare_v2(db, sql, -1, &stmt, 0) == SQLITE_OK) {
                sqlite3_bind_text(stmt, 1, soignant.c_str(), -1, SQLITE_STATIC);
                sqlite3_bind_int(stmt, 2, alerte_id);
                sqlite3_step(stmt);
                sqlite3_finalize(stmt);

                log("INFO", "Alerte " + std::to_string(alerte_id) +
                    " prise par " + soignant);

                return crow::response(200, "{\"status\":\"ok\"}");
            }

            return crow::response(500);
        });

        // Servir interface web
        CROW_ROUTE(app, "/")([]() {
            crow::mustache::context ctx;
            return crow::mustache::load("index.html").render(ctx);
        });

        // Démarrer serveur
        std::cout << "Serveur démarré sur port " << port << std::endl;
        app.port(port).multithreaded().run();
    }

private:
    void broadcastWebSocket(int chambre, int besoin, int priorite) {
        // Implémentation WebSocket simple
        // (À compléter avec Crow WebSocket)
    }
};

int main() {
    NurseCallServer server;
    server.run();
    return 0;
}
```