Серверная часть

Библиотека SQL

cpp

#include <sqlite3.h>

#include <iostream>

int main() {

sqlite3\* db;

int rc = sqlite3\_open("data/chat.db", &db);

if (rc) {

std::cerr << "Can't open database: " << sqlite3\_errmsg(db) << std::endl;

return rc;

}

// Создание таблицы пользователей

const char\* create\_users\_sql = "CREATE TABLE IF NOT EXISTS Users ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

"username TEXT NOT NULL, "

"password TEXT NOT NULL);";

// Выполнение SQL-запроса

sqlite3\_exec(db, create\_users\_sql, 0, 0, 0);

// Создание таблицы сообщений

const char\* create\_messages\_sql = "CREATE TABLE IF NOT EXISTS Messages ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

"sender\_id INTEGER NOT NULL, "

"receiver\_id INTEGER NOT NULL, "

"message TEXT NOT NULL, "

"timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP, "

"FOREIGN KEY (sender\_id) REFERENCES Users(id), "

"FOREIGN KEY (receiver\_id) REFERENCES Users(id));";

// Выполнение SQL-запроса

sqlite3\_exec(db, create\_messages\_sql, 0, 0, 0);

sqlite3\_close(db);

return 0;

}

Сетевое взаимодействие

main

Сервер

cpp

#include <boost/asio.hpp>

#include <iostream>

#include <thread>

using boost::asio::ip::tcp;

// Обработка сессии взаимодействия с клиентом

void session(tcp::socket sock) {

try {

while (true) {

char data[512]; // Буфер для данных

boost::system::error\_code error;

// Чтение данных от клиента

size\_t length = sock.read\_some(boost::asio::buffer(data), error);

// Обработка ошибок

if (error == boost::asio::error::eof)

break; // Соединение закрыто клиентом

else if (error)

throw boost::system::system\_error(error);

// Отправка данных обратно клиенту (эхо)

boost::asio::write(sock, boost::asio::buffer(data, length));

}

} catch (std::exception& e) {

std::cerr << "Exception: " << e.what() << std::endl;

}

}

int main() {

try {

// Создание объекта io\_context для управления асинхронными операциями

boost::asio::io\_context io\_context;

// Создание TCP-акцептора для прослушивания входящих соединений на порту 12345

tcp::acceptor acceptor(io\_context, tcp::endpoint(tcp::v4(), 12345));

// Основной цикл сервера

while (true) {

// Создание сокета для взаимодействия с клиентом

tcp::socket socket(io\_context);

// Ожидание подключения клиента

acceptor.accept(socket);

// Запуск новой сессии в отдельном потоке для обработки клиента

std::thread(session, std::move(socket)).detach();

}

} catch (std::exception& e) {

std::cerr << "Exception: " << e.what() << std::endl;

}

return 0;

}

Создание контейнера

dockerfile

FROM gcc:latest

# Установка зависимостей

RUN apt-get update && apt-get install -y libboost-all-dev libsqlite3-dev

# Копирование исходного кода в контейнер

COPY . /usr/src/chat\_server

# Сборка проекта

WORKDIR /usr/src/chat\_server

RUN g++ -o chat\_server main.cpp -lboost\_system -lpthread -lsqlite3

# Запуск сервера

CMD ["./chat\_server"]

Запуск контейнера

yaml

version: '3'

services:

chat\_server:

build: .

ports:

- "12345:12345"

volumes:

- chat\_data:/usr/src/chat\_server/data # Подключаем том

depends\_on:

- db

db:

image: mysql:latest

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: chat\_db

ports:

- "3306:3306"

volumes:

chat\_data: # Создаем том

Проверка единственного подключения

cpp

class Database {

public:

static Database& getInstance() {

static Database instance; // Инициализация единственного экземпляра

return instance;

}

// Другие методы для работы с базой данных...

private:

Database() { /\* Инициализация подключения \*/ }

~Database() { /\* Закрытие подключения \*/ }

Database(const Database&) = delete;

Database& operator=(const Database&) = delete;

};

Подключение и отправка сообщения

cpp

class Controller {

public:

void registerUser(const std::string& username, const std::string& password) {

// Взаимодействие с моделью для регистрации пользователя

model.registerUser(username, password);

}

void sendMessage(const std::string& message) {

// Обработка отправки сообщения

model.storeMessage(message);

view.updateMessageList();

}

};

Асинхронное чтение

cpp

void asyncRead(tcp::socket& socket) {

socket.async\_read\_some(boost::asio::buffer(data),

[this](const boost::system::error\_code& error, std::size\_t bytes\_transferred) {

if (!error) {

// Обработка полученных данных

asyncRead(socket); // Продолжаем чтение

}

});

}

Логирование:

1. Событие подключения

cpp

void logUserConnection(const std::string& username) {

std::ofstream logFile("server.log", std::ios\_base::app);

logFile << "User connected: " << username << " at " << getCurrentTime() << std::endl;

}

1. Ошибка

cpp

void logError(const std::string& errorMessage) {

std::ofstream logFile("server.log", std::ios\_base::app);

logFile << "Error: " << errorMessage << " at " << getCurrentTime() << std::endl;

}

1. Запрос к базе

cpp

void logDatabaseQuery(const std::string& query) {

std::ofstream logFile("server.log", std::ios\_base::app);

logFile << "Database query executed: " << query << " at " << getCurrentTime() << std::endl;

}

1. Время обработки

cpp

#include <chrono>

void someDatabaseOperation() {

auto start = std::chrono::high\_resolution\_clock::now();

// Выполнение операции с базой данных

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> duration = end - start;

logDatabaseQuery("Operation took " + std::to\_string(duration.count()) + " ms");

}

Пример транзакции

cpp

sqlite3\_exec(db, "BEGIN TRANSACTION;", nullptr, nullptr, &errMsg);

// Выполнение нескольких SQL-запросов

sqlite3\_exec(db, "INSERT INTO Messages (sender\_id, receiver\_id, message) VALUES (1, 2, 'Hello');", nullptr, nullptr, &errMsg);

sqlite3\_exec(db, "INSERT INTO Messages (sender\_id, receiver\_id, message) VALUES (2, 1, 'Hi!');", nullptr, nullptr, &errMsg);

sqlite3\_exec(db, "COMMIT;", nullptr, nullptr, &errMsg);