**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**

**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Веб-технології та веб-дизайн**

**Лабораторна робота №11**

**на тему: «Підключення БД»**

Виконав: студент 3 курсу, групи ІПЗ-3.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Перевірив\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Рябов Д.М.

**Одеса  2024**

**ЗАВДАННЯ**

Подключить базу данных к запрограммированному ранее интерфейсу

**ВИКОНАННЯ**

**main.cpp**

#include "config.hpp"

#include "DBQuery.hpp"

#include "Logger.hpp"

#include "ServerStarterController.hpp"

#include "ServerStarterModel.hpp"

#include <cinttypes>

#include <chrono>

#include <ctime>

#include <iomanip>

#include <sstream>

#include <thread>

#include <memory>

#include <queue>

#include <sys/stat.h> // mkdir;

namespace db = server::db;

std::string current\_date\_time\_as\_string() {

// Get the current time using std::chrono

auto now = std::chrono::system\_clock::now();

// Convert the current time to a time\_t object

std::time\_t now\_c = std::chrono::system\_clock::to\_time\_t(now);

// Convert time\_t to a struct tm (broken down time)

std::tm tm\_struct = \*std::localtime(&now\_c);

// Format the date-time as a string

std::stringstream ss;

ss << std::put\_time(&tm\_struct, "%Y.%m.%d\_%H.%M.%S");

return ss.str();

}

std::string generate\_log\_file\_name(const std::string &first\_name\_part) {

std::stringstream file\_name\_stream;

file\_name\_stream << first\_name\_part << "\_" << current\_date\_time\_as\_string() << ".log";

return std::move(file\_name\_stream.str());

}

int main(int argc, char \*\*argv) {

const std::string LOGS\_FOLDER\_DIR\_NAME = "logs";

mkdir(LOGS\_FOLDER\_DIR\_NAME.c\_str(), 0755);

BLOG\_INIT(std::move(generate\_log\_file\_name(LOGS\_FOLDER\_DIR\_NAME + "/server\_logs.txt")), true);

BDECLARE\_TAG\_SCOPE("", \_\_FUNCTION\_\_);

std::queue<std::thread> thread\_pull;

auto server\_starter\_model = std::make\_shared<server::serverstarter::models::ServerStarterModel>();

server::serverstarter::controllers::ServerStarterController server\_starter\_controller(server\_starter\_model);

db::DBQuery dbquery;

thread\_pull.push(std::thread(&server::serverstarter::controllers::ServerStarterController::start, &server\_starter\_controller));

dbquery.output\_all\_users();

while (!thread\_pull.empty()) {

if (thread\_pull.front().joinable()) {

thread\_pull.front().join();

}

thread\_pull.pop();

}

return 0;

}

**DBQuery.hpp**

#ifndef POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP

#define POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP

#include <pqxx/connection>

#include <pqxx/transaction>

#include <memory>

namespace server::db {

class DBQuery {

public:

DBQuery(void);

DBQuery(const DBQuery &other) = delete;

~DBQuery(void);

void operator=(const DBQuery &other) = delete;

void output\_all\_users(void);

private:

std::unique\_ptr<pqxx::connection> m\_db\_connection;

std::unique\_ptr<pqxx::transaction\_base> m\_db\_transaction;

};

} // !server::db;

#endif // !POLYCLINIC\_SERVER\_DB\_DBQUERY\_HPP;

**DBQuery.cpp**

#include "DBQuery.hpp"

#include "config.hpp"

#include "Logger.hpp"

#include <pqxx/result>

#include <sstream>

#include <thread>

namespace config = server::common::config;

namespace server::db

{

DBQuery::DBQuery() {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

std::stringstream db\_connection\_command\_stream;

db\_connection\_command\_stream << "host=" << config::DB\_HOST << " port=" << config::DB\_PORT

<< " user=" << config::DB\_USERNAME << " password=" << config::DB\_PASSWORD

<< " dbname=" << config::DB\_NAME;

BLOG\_INFO("constructor called on thread #", std::this\_thread::get\_id());

BLOG\_INFO("Make connection to db. ", db\_connection\_command\_stream.str());

m\_db\_connection = std::make\_unique<pqxx::connection>(db\_connection\_command\_stream.str());

if (m\_db\_connection->is\_open()) {

BLOG\_DEBUG("Connection success");

} else {

BLOG\_FATAL("Connection failed");

abort();

}

m\_db\_transaction = std::make\_unique<pqxx::work>(\*m\_db\_connection);

};

DBQuery::~DBQuery(void) {

m\_db\_connection->close();

}

void DBQuery::output\_all\_users(void) {

BDECLARE\_TAG\_SCOPE("DBQuery", \_\_FUNCTION\_\_);

BLOG\_INFO("called");

pqxx::result res = m\_db\_transaction->exec("SELECT \* FROM users");

m\_db\_transaction->commit();

for (const auto &row : res) {

std::stringstream table\_info\_output;

for (const auto &field : row) {

table\_info\_output << field.c\_str() << '\t';

}

BLOG\_DEBUG(table\_info\_output.str());

}

}

} // !server::db;

**config.hpp**

#ifndef POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP

#define POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP

#include <string>

#include <cinttypes>

namespace server::common::config {

const std::string LOCALHOST {"127.0.0.1"};

const std::string DB\_HOST {LOCALHOST};

const std::string DB\_USERNAME {"mbukhta"};

const std::string DB\_PASSWORD {"1qa@WS3ed"};

const std::string DB\_NAME {"polyclinic"};

const std::uint16\_t DB\_PORT {5432};

const std::uint16\_t SERVER\_PORT {8081};

const std::uint8\_t SERVER\_LISTEN\_SIZE = 10;

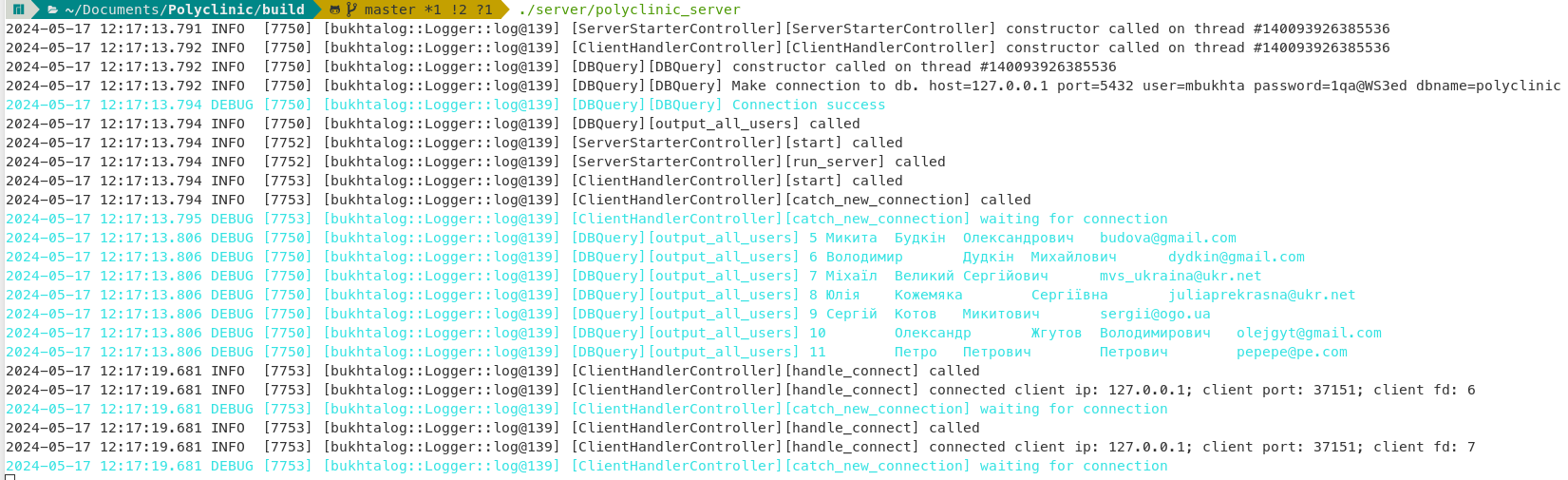
const char STR\_EOF = '\0';

} // !server::common::config;

#endif // !POLYCLINIC\_SERVER\_COMMON\_CONFIG\_HPP;

**РЕЗУЛЬТАТ ВИКОНАННЯ**

Рисунок 1 – результат виконання.



**ВИСНОВОК**

У ході виконання лабораторної роботи на тему "Підключення бази даних" було детально розглянуто процес підключення бази даних PostgreSQL до серверу, написаного мовою програмування C++. Для досягнення цієї мети було використано бібліотеку libpqxx, яка є офіційним інтерфейсом для взаємодії з PostgreSQL у C++.

Під час лабораторної роботи ми ознайомилися з основними принципами роботи з базами даних: встановлення з'єднання з базою даних, виконання SQL-запитів, отримання та обробка результатів. Також ми навчились обробляти можливі помилки, що можуть виникнути під час з'єднання або виконання запитів, що є критично важливим для побудови надійних програм.

Практичне виконання роботи дало змогу зрозуміти архітектуру взаємодії між сервером та базою даних, а також усвідомити важливість правильного управління ресурсами та безпекою при роботі з базами даних.

Таким чином, результати цієї лабораторної роботи заклали фундаментальні знання, необхідні для створення ефективних та безпечних серверних додатків, які можуть взаємодіяти з базами даних, що є важливою складовою у розробці сучасних програмних продуктів.