Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

«Захист програмного забезпечення»

Розрахунково-графічна робота

Виконав:
студент групи ФБ-22
Пунько А. Ю.

Київ – 2025

1. ЦІЛЬ розрахунково-графічних робіт

Метою розрахунково-графічної роботи є освоєння методів автентифікції на підставі біометричних характеристик користувача.

2. ЗАВДАННЯ НА розрахунково-графічних робіт

Розробити програму автентифікації користувача по клавіатурному почерку.

Вимоги до програми:

1. Програма повинна працювати в двох режимах:
   - навчання (створення біометричного еталону)
   - ідентифікації (порівняння з біометричним еталоном).
2. На етапі навчання необхідно визначати еталонні статистичні параметри клавіатурного почерку – оцінки математичного чекання і дисперсії тривалості утримання клавіш. Параметри повинні записуватися у файл. Вчення виробляти по багатократному набору фіксованій контрольній фразі, символи якої рівномірно розподілені по клавіатурі.
3. На етапі ідентифікації необхідно визначити параметри введеної контрольної фрази і перевірити гіпотезу про те, що отримані оцінки математичного очікування і дисперсії належать тому ж розподілу, що і параметри біометричного еталону. На цьому етапі необхідно відображувати отримані оцінки і еталонні параметри. Рівень значущості критерію задавати в діалоговому вікні.
4. На етапах навчання і ідентифікації передбачити можливість відбракування грубих помилок (окремих вимірів)

3. Вимоги до оформлення роботи:

Оформлена робота повинна містити:

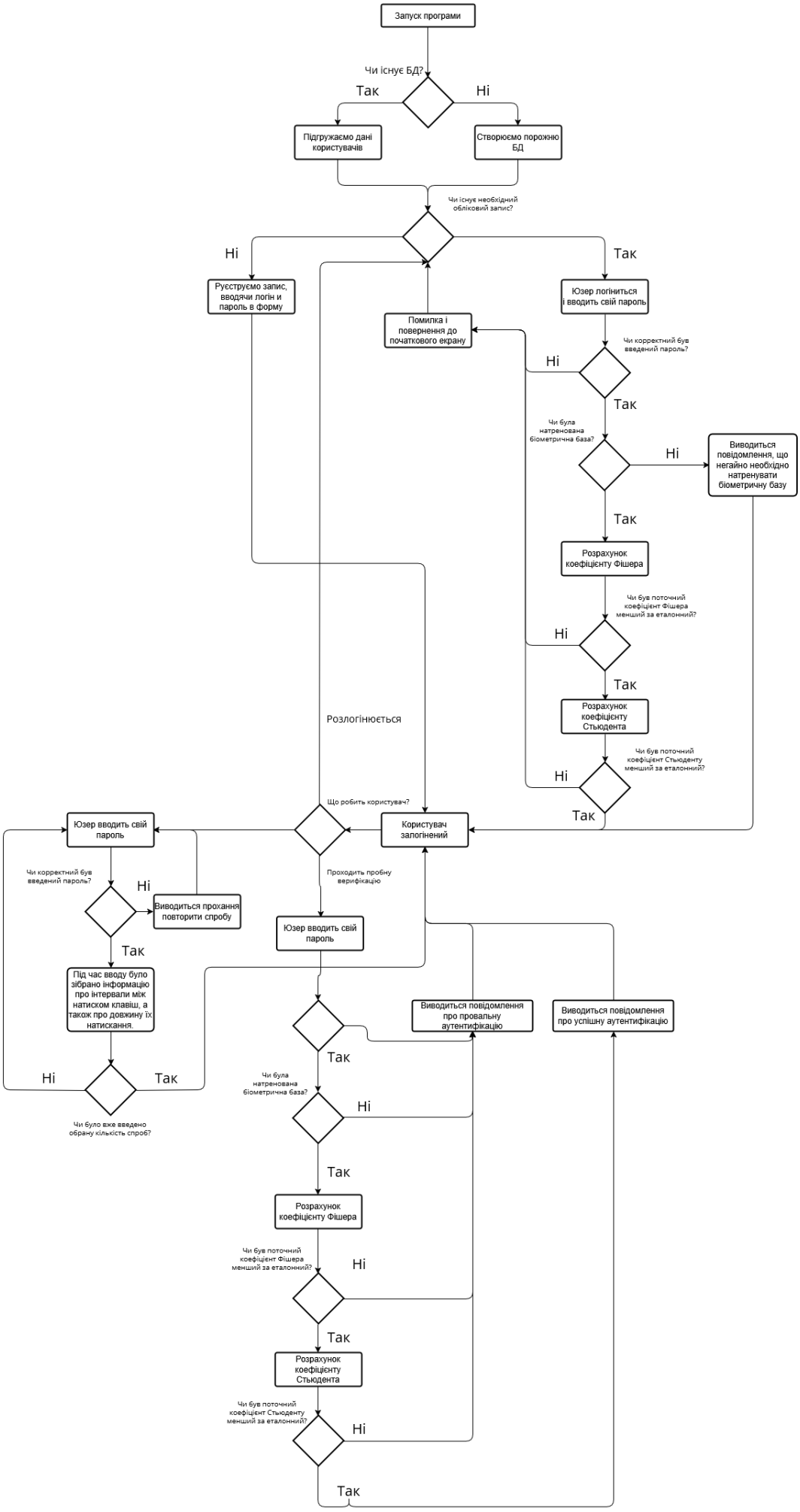
- постановку завдання
- опис і блок-схему алгоритму її рішення
- лістинг функціональної частини програми
- результати експериментальних досліджень (помилки першого і другого роду для різних осіб)

Необхідно також продемонструвати робочу версію додатку.

I. ПОСТАНОВКА ЗАВДАННЯ І ОПИС ЙОГО ВИРІШЕННЯ

Завданням є розроблення певного алгоритму, який відповідає за зчитування клавіатурного «почерку» користувача комп'ютера, щоб інша людина, з відповідно іншим «почерком», навіть знаючи пароль, не могла без проблем увійти до чужого облікового запису. Звісно, є шанс співпадіння почерку, навіть при строгому відборі, бо універсального захисту від всіх і всього, очевидно, не існує, але це все одно дуже сильно зменшує шанс неавторизованого доступу.

Мовою реалізації була C++17, основою для GUI була бібліотека wxWidgets (якщо якісь моменти в реалізації GUI в цій бібліотеці були неоптимальні, то можу лише сказати, що це мій перший досвід роботи з цією бібліотекою, раніше я працював найбільше з WinForms), в розрахунках допомагала бібліотека Boost.Math.
Програма має БД, що зберігається в файлі "db.json". Там зберігаються ім'я користувача, пароль, а також еталоні зразки інтервалів між натисканнями клавіш, а також довжина натискань. Також для кожного користувача є змога індивідуально підібрати кількість спроб у режимі тренування (5, 10, 15, 20), а також і строгість відбору (від 0,01 до 0,99, хоча і очевидно, що із значеннями більше 0,15, скоріш за все, навіть власнику аккаунта буде дуже проблематично зайти у свій обліковий запис, особисто у мене викликалися проблеми вже при 0,1).
Користувач може зареєструвати новий обліковий запис, але в такому випадку йому потрібно буде додатково натренувати свою біометричну базу, інакше доступ до його аккаунту буде доступний просто по паролю, без біометричних тестів, про що людині буде нагадано кожний раз, коли вона успішно логіниться до аккаунту. В разі успішного логіну буде надано повний функціонал програми: режим тренування і режим пробної верифікації, а також доступ до індивідуальних налаштувань, про які я говорив вище.
В режимі тренування людина пройде обрану кількість спроб вводу паролю (п'ять по замовченням), після цього натреновану базу буде завантажено до БД.
В режимі пробної аутентифікації людина має спробу пройти справжню верифікацію свого вводу паролю, щоб мати можливість обрати комфортну для себе строгість біометричного тесту (наприклад, для мене це значення від 0,03 до 0,07).


Запуск програми
Чи існує БД?
Так
Ні
Підгружаємо дані користувачів
Створюємо порожню БД
Чи існує необхідний обліковий запис?
Ні
Так
Руєструємо запис, вводячи логін и пароль в форму
Помилка і повернення до початкового екрану
Юзер логіниться і вводить свій пароль
Чи корректний був введений пароль?
Ні
Так
Чи була натренована біометрична база?
Ні
Виводиться повідомлення, що негайно необхідно натренувати біометричну базу
Так
Розрахунок коефіцієнту Фішера
Чи був поточний коефіцієнт Фішера менший за еталонний?
Ні
Так
Розрахунок коефіцієнту Стьюдента
Чи був поточний коефіцієнт Стьюденту менший за еталонний?
Ні
Так
Розлогінюється
Що робить користувач?
Користувач залогінений
Юзер вводить свій пароль
Проходить пробну верифікацію
Чи корректний був введений пароль?
Ні
Виводиться прохання повторити спробу
Так
Під час вводу було зібрано інформацію про інтервали між натиском клавіш, а також про довжину їх натискання.
Ні
Так
Чи було вже введено обрану кількість спроб?
Юзер вводить свій пароль
Виводиться повідомлення про провальну аутентифікацію
Виводиться повідомлення про успішну аутентифікацію
Так
Чи була натренована біометрична база?
Ні
Так
Розрахунок коефіцієнту Фішера
Чи був поточний коефіцієнт Фішера менший за еталонний?
Ні
Так
Розрахунок коефіцієнту Стьюдента
Чи був поточний коефіцієнт Стьюденту менший за еталонний?
Ні
Так

II. ЛІСТИНГ ФУНКЦІОНАЛЬНОЇ ЧАСТИНИ ПРОГРАМИ

```
class MainForm : public wxFrame {
public:
    MainForm();
    ~MainForm() override;

private:
    enum class Mode { Idle, Training, Verification, TrialVerification };
    enum class PulseType { Success, Error };

    void buildUI();
    void loggedState();
    void loggedoutState();
    void textboxEnable();
    void textboxDisable();

    void loadDB();
    void saveDB();
    void saveBioprint();

    void onKeyDown(wxKeyEvent& event);
    void onKeyUp(wxKeyEvent& event);
    void onEnter(wxCommandEvent& event);
    void OnBlinkTimer(wxTimerEvent& event);

    void startTraining(wxCommandEvent& event);
    void startTrialVerification(wxCommandEvent& event);
    void registerUser(wxCommandEvent& event);
    void login(wxCommandEvent& event);
    void logout(wxCommandEvent& event);
    void processAttempt();

    bool verifyBio();
    bool checkSample(const std::vector<double>& sample, const std::vector<double>& ethalon);
    std::vector<double> filterErrors(const std::vector<double>& data);
    double calculateVariance(const std::vector<double>& data);

    void BackgroundPulse(PulseType type);

    static constexpr unsigned short WIDGET_WIDTH = 480;
    static constexpr unsigned short WIDGET_HEIGHT = 320;
```

```
    static constexpr unsigned short WIDGET_HEIGHT = 320;
    static constexpr int blinkIntervalMs = 30;
    static constexpr int blinkDurationMs = 1000;

    wxAuiManager auiManager;
    wxTimer blinkTimer{ this, 3000 };
    wxColour origBgColor;
    std::chrono::steady_clock::time_point blinkStart;

    nlohmann::json db;
    std::filesystem::path dataDir;
    std::filesystem::path DBdir;

    std::string currUser;
    std::string currPassword;
    unsigned char currAttempts;
    double userAlpha;

    Mode currentMode{ Mode::Idle };
    PulseType currentPulseType{ PulseType::Success };
    int attempts{ 0 };

    std::map<int, std::chrono::high_resolution_clock::time_point> pressTime;
    std::vector<double> currHolds;
    std::vector<double> currIntervals;
    std::chrono::high_resolution_clock::time_point lastRelease;
    std::vector<std::vector<double>> holdTimes;
    std::vector<std::vector<double>> intervalTimes;

    wxPanel* mainPanel{ nullptr };
    wxMenu* optionsMenu{ nullptr };
    wxMenu* attemptsMenu{ nullptr };
    wxTextCtrl* textBox{ nullptr };
    wxButton* buttonEnter{ nullptr };
    wxButton* buttonTrain{ nullptr };
    wxButton* buttonVerification{ nullptr };
    wxButton* buttonRegister{ nullptr };
    wxButton* buttonLogout{ nullptr };
    wxButton* buttonLogin{ nullptr };
    wxStaticText* usernameLabel{ nullptr };

    wxDECLARE_EVENT_TABLE();
};
```

Так виглядає прототип класу програми, але розглядати роботу з GUI, банальний ввід-вивід в json файл і так далі я не бачу сенсу, бо це не торкається тематики РГР, тому червоними стрілками я показав на методи, які, на мою думку, має сенс розгледіти, бо саме вони торкають основної логіки програми.

```
wxBEGIN_EVENT_TABLE(MainForm, wxFrame)
    EVT_CHAR_HOOK(MainForm::onKeyDown)
    EVT_CHAR(MainForm::onKeyUp)
    EVT_TEXT_ENTER(wxID_ANY, MainForm::onEnter)
    EVT_BUTTON(wxID_OK, MainForm::onEnter)
    EVT_TIMER(3000, MainForm::OnBlinkTimer)
wxEND_EVENT_TABLE()
```

Також хочу показати таблицю івентів, бо вона має сенс в бібліотеці wxWidgets як певний зручний функціонал для розташування певних хуків в програмі.

В ній я повісив методи «onKeyDown» на натискання клавіши, «onKeyUp», відповідно, за відпускання клавіші, на підтвердження вводу тексту через графічну кнопку було повішено метод «onEnter», і на підтвердження вводу через клавішу «Enter» також було повішено цей же метод.

```
void MainForm::onKeyDown(wxKeyEvent& event) {
    if (currentMode == Mode::Idle) { event.Skip(); return; }

    int kc = event.GetKeyCode();
    auto now = std::chrono::high_resolution_clock::now();
    pressTime[kc] = now;

    if (lastRelease.time_since_epoch().count() > 0) {
        currIntervals.push_back(std::chrono::duration<double>(now - lastRelease).count());
    }
    event.Skip();
}

void MainForm::onKeyUp(wxKeyEvent& event) {
    if (currentMode == Mode::Idle) { event.Skip(); return; }

    int kc = event.GetKeyCode();
    auto now = std::chrono::high_resolution_clock::now();
    if (auto it = pressTime.find(kc); it != pressTime.end()) {
        currHolds.push_back(std::chrono::duration<double>(now - it->second).count());
        pressTime.erase(it);
    }
    lastRelease = now;
    event.Skip();
}
```

Методи «onKeyUp» і «onKeyDown» не мають нічого складного, вони лише відловлюють натискання клавіш при вводі тексту на клавіатурі. В «onKeyDown» таймінги натиснутих клавіш поміщаються у вектор pressTime, а також розраховуються інтервали натискання клавіш (якщо це не перша клавіша). А у «onKeyUp» розраховуються час натискання, змінюється змінна, яка відповідає на останнє відпускання клавіши і розраховуються таймінги утримання клавіш.

Програма, якщо можна так сказати, має 4 режими. Вони необхідні для розуміння перевірки `if (currentMode == Mode::Idle)`, а також для розуміння методу «onEnter», який ми розглянемо наступним. Оси ці режими:

1. Idle – програма не відловлює нічого. Це необхідно для коректної роботи хуку, який, інакше, відловлював би завжди натискання по клавіатурі, а не лише під час вводу, що, очевидно, дуже погано впливає на коректність біометричної верифікації.
2. Training – режим тренування біометричного еталону
3. Verification – режим справжньої верифікації
4. TrialVerification – режим пробної верифікації.

```
void MainForm::onEnter(wxCommandEvent& event) {
    if (currentMode == Mode::Idle) {
        return;
    }

    wxString typed = textBox->GetValue();
    textBox->Clear();
    textboxDisable();

    switch (currentMode) {
        case Mode::Training:
            if (typed.ToStdString() != currPassword) {
                wxMessageBox("Incorrect password. Please try again!", "Error");
                currHolds.clear();
                currIntervals.clear();
                pressTime.clear();
                lastRelease = {};
                textboxEnable();
                return;
            }

            processAttempt();

            if (attempts >= currAttempts) {
                saveBioprint();
                wxMessageBox("Training complete!", "Info");
                currentMode = Mode::Idle;
            }
            else {
                wxMessageBox(wxString::Format("Attempt %d/%d saved. Next attempt:", attempts, currAttempts), "Training");
                currHolds.clear();
                currIntervals.clear();
                pressTime.clear();
```

```
            else {
                wxMessageBox(wxString::Format("Attempt %d/%d saved. Next attempt:", attempts, currAttempts), "Training");
                currHolds.clear();
                currIntervals.clear();
                pressTime.clear();
                lastRelease = {};
                textboxEnable();
            }
            return;
        case Mode::Verification:
            currentMode = Mode::Idle;

            if (typed.ToStdString() != currPassword) {
                BackgroundPulse(PulseType::Error);
                wxMessageBox("Incorrect password.", "Error");
                currUser.clear();
                return;
            }

            if (!verifyBio()) {
                BackgroundPulse(PulseType::Error);
                wxMessageBox("Biometric mismatch. Access denied.", "Error");
                currUser.clear();
                return;
            }

            BackgroundPulse(PulseType::Success);
            wxMessageBox("Welcome, " + wxString(currUser), "Access granted");
            loggedState();
            return;
        case Mode::TrialVerification:
            if (typed.ToStdString() != currPassword) {
                BackgroundPulse(PulseType::Error);
                wxMessageBox("Incorrect password. Verification failed!", "Error");
            }
            else {
```

```
            return;
        case Mode::TrialVerification:
            if (typed.ToStdString() != currPassword) {
                BackgroundPulse(PulseType::Error);
                wxMessageBox("Incorrect password. Verification failed!", "Error");
            }
            else {
                if (!verifyBio()) {
                    BackgroundPulse(PulseType::Error);
                    wxMessageBox("You failed verification!", "Error");
                }
                else {
                    BackgroundPulse(PulseType::Success);
                    wxMessageBox("You succeeded verification!", "Success");
                }
            }
            currentMode = Mode::Idle;
        default: return;
    }
}
```

Відповідно в «onEnter» ми бачимо результат опрацювання вводу залежно від режиму.

В режимі тренування ми вводимо строчку, перевіряємо співпадіння з паролем і оброблюємо спробу вводу (метод processAttempt буде розглянуто пізніше). Далі, залежно від кількості успішних спроб вводу паролю, ми або завершуємо тренування, або виводимо кількість пройдених спроб.

В режимі верифікації ми перевіряємо правильність вводу паролю і чи співпадає біометричний відбиток. Якщо так – користувача допущено до облікового запису, якщо ж щось з цього йде не так – його не допускають до аккаунту.

В режимі пробної верифікації все так само, але нема загрози викидання користувача з облікового запису при невдачі, лише успішність або неуспішність верифікації.

```
void MainForm::processAttempt() {
    auto fh = filterErrors(currHolds);
    auto fi = filterErrors(currIntervals);

    holdTimes.push_back(fh);
    intervalTimes.push_back(fi);
    attempts++;

    currHolds.clear();
    currIntervals.clear();
}
```

Так виглядає метод, що обробляє кожну спробу в тренуванні еталону. Він відфільтровує помилки інтервалів і утримувань і поміщає результат в еталон користувача, а також збільшує поточну спробу.

```cpp
std::vector<double> MainForm::filterErrors(const std::vector<double>& data) {
    std::vector<double> filtered(data);

    if (filtered.size() < 3)
        return filtered;

    bool removed;
    do {
        removed = false;

        if (filtered.size() < 3)
            break;

        for (int i = 0; i < static_cast<int>(filtered.size()); ++i) {
            if (filtered.size() - 1 < 2)
                continue;

            double sum = std::accumulate(filtered.begin(), filtered.end(), 0.0) - filtered[i];
            double mI = sum / (filtered.size() - 1);

            double sI2 = 0.0;
            for (int j = 0; j < static_cast<int>(filtered.size()); ++j) {
                if (j == i) continue;
                sI2 += std::pow(filtered[j] - mI, 2);
            }

            if (filtered.size() - 2 <= 0)
                continue;
            sI2 /= (filtered.size() - 2);

            double sI = std::sqrt(sI2);
            if (sI == 0.0)
                continue;

            boost::math::students_t studCoef(filtered.size() - 2);
            double tT = boost::math::quantile(studCoef, 0.975);

            double tP = std::abs((filtered[i] - mI) / sI);
            if (tP > tT) {
                filtered.erase(filtered.begin() + i);
                removed = true;
                break;
            }
        }
    } while (removed);

    return filtered;
}
```

Так виглядає метод обробки помилок. Якщо даних менше аніж 3 – обробляти безсенсово. Далі йде цикл, який вже обробляє помилки, який буде активний, поки є хоча б одне видалення елементу за ітерацію.

Якщо розмір вектора в результаті очистки стане менший за 3 – одразу перериваємо цикл и повертаємо результат. Далі ми рахуємо матсподівання вектору даних без одного елементу «і». Далі ми обраховуємо дисперсію для отриманого вектору, знову ж таки без елементу «і» і отримуємо середньоквадратичне відхилення. Далі отримується коефіцієнт Стьюдента для степені свободи, який дорівнює розміру вектор – 2, і для рівня значимості 2.5% (зробив відрахування трохи м’якішим, бо в іншому випадку маленькі паролі можуть майже повністю бути відфільтровані). Далі розраховується і коефіцієнт

Стьюдента і для отриманої множини, і якщо він більший за табличний – елемент «і» відкидається.

```
double MainForm::calculateVariance(const std::vector<double>& data) {
    if (data.empty() || data.size() == 1) return 0.0;

    double mean = std::accumulate(data.begin(), data.end(), 0.0) / data.size();

    double sumSquaredDiffs = 0.0;
    for (const auto& x : data) {
        sumSquaredDiffs += std::pow((x - mean), 2);
    }
    return sumSquaredDiffs / (data.size() - 1);
}
```

Метод, який обраховує дисперсію. Тут обраховується матсподіванна, а далі суммуються елементи, від яких віднімається матсподівання, і ця різниця піднімається до квадрату. В кінці повертається ця сума поділена на розмір масиву - 1.

```
bool MainForm::verifyBio() {
    if (!db["users"].contains(currUser)) return false;

    const auto& user = db["users"][currUser];
    const auto& samples = user["samples"];

    if (!samples.contains("holds") || samples["holds"].empty()) {
        wxMessageBox("You logged without biometrical verification because it is empty. Please, complete training", "ALERT!!!");
        return true;
    }

    std::vector<double> intervals;
    for (const auto& sample : samples["intervals"]) {
        for (const auto& value : sample) {
            intervals.push_back(value.get<double>());
        }
    }

    std::vector<double> holds;
    for (const auto& sample : samples["holds"]) {
        for (const auto& value : sample) {
            holds.push_back(value.get<double>());
        }
    }

    if (!checkSample(intervals, currIntervals) || !checkSample(holds, currHolds)) return false;

    return true;
}
```

Так виглядає метод, що відповідає за верифікацію біометрії. З БД дістаються еталонні заміри, які потім передаються у метод «checkSample». Я перевіряю як і інтервали між натисками клавіш, так і утримання клавіш.

```cpp
bool MainForm::checkSample(const std::vector<double>& sample, const std::vector<double>& ethalon) {
    auto filteredEthalon = filterErrors(ethalon);

    if (sample.size() < 2 || filteredEthalon.size() < 2) {
        wxMessageBox("Not enough data points to perform statistical tests. Skipping check.", "Warning");
        return true;
    }

    double s1 = calculateVariance(sample);
    double s2 = calculateVariance(filteredEthalon);

    double Fp = (s1 < s2 ? s2 / s1 : s1 / s2);
    double Ft = 0;
    double pF = 1.0 - userAlpha;
    if (pF <= 0.0 || pF >= 1.0) pF = 0.95;

    try {
        boost::math::fisher_f_distribution<> fDist(sample.size() - 1, filteredEthalon.size() - 1);
        Ft = boost::math::quantile(fDist, pF);
    }
    catch (const std::domain_error& ex) {
        wxMessageBox("F-test skipped: " + wxString(ex.what()), "F-Test Error", wxICON_ERROR);
        return true;
    }

    wxMessageBox(
        wxString::Format("F_p = %f.5\nF_T = %f.5\nVerdict is %s", Fp, Ft, (Fp > Ft) ? "NEGATIVE" : "POSITIVE"),
        "F-Test"
    );
    if (Fp > Ft) return false;

    double My = std::accumulate(filteredEthalon.begin(), filteredEthalon.end(), 0.0) / filteredEthalon.size();
    double Mx = std::accumulate(sample.begin(), sample.end(), 0.0) / sample.size();

    double S2y = 0;
    for (double v : filteredEthalon) S2y += (v - My) * (v - My);
```

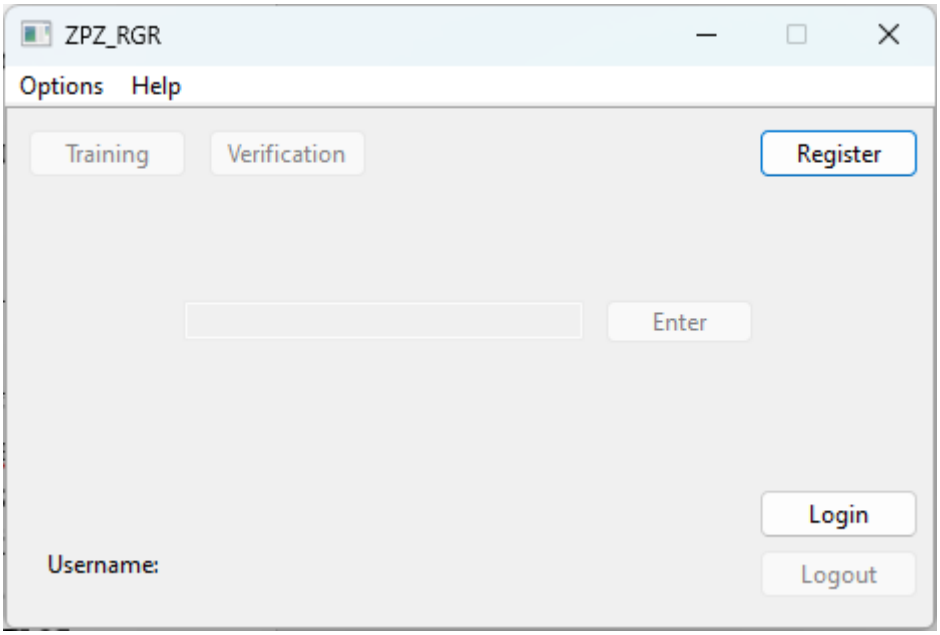
```cpp
    double My = std::accumulate(filteredEthalon.begin(), filteredEthalon.end(), 0.0) / filteredEthalon.size();
    double Mx = std::accumulate(sample.begin(), sample.end(), 0.0) / sample.size();

    double S2y = 0;
    for (double v : filteredEthalon) S2y += std::pow((v - My), 2);
    S2y /= (filteredEthalon.size() - 1);

    double S2x = 0;
    for (double v : sample) S2x += std::pow((v - Mx), 2);
    S2x /= (sample.size() - 1);

    double pooledVar = ((filteredEthalon.size() - 1) * S2y + (sample.size() - 1) * S2x)
        / (filteredEthalon.size() + sample.size() - 2);
    double Sp = std::sqrt(pooledVar);
    double tStat = (Mx - My) / (Sp * std::sqrt(1.0 / filteredEthalon.size() + 1.0 / sample.size()));

    std::size_t dfT = filteredEthalon.size() + sample.size() - 2;
    double tCrit = 0;
    double pT = 1.0 - userAlpha;
    if (pT <= 0.0 || pT >= 1.0) pT = 0.975;

    try {
        boost::math::students_t tDist(dfT);
        tCrit = boost::math::quantile(tDist, pT);
    }
    catch (const std::domain_error& ex) {
        wxMessageBox("T-test skipped: " + wxString(ex.what()), "T-Test Error", wxICON_ERROR);
        return true;
    }

    wxMessageBox(
        wxString::Format("t_stat = %f.5\ntCritical = %f.5\nVerdict is %s", std::abs(tStat), tCrit,
            (std::abs(tStat) > tCrit) ? "NEGATIVE" : "POSITIVE"),
        "T-Test"
    );

    if (std::abs(tStat) > tCrit) return false;

    return true;
```

Метод «checkSample» доволі великий. Спочатку ми розраховуємо коефіцієнт Фішера, а далі коефіцієнт Стьюдента.

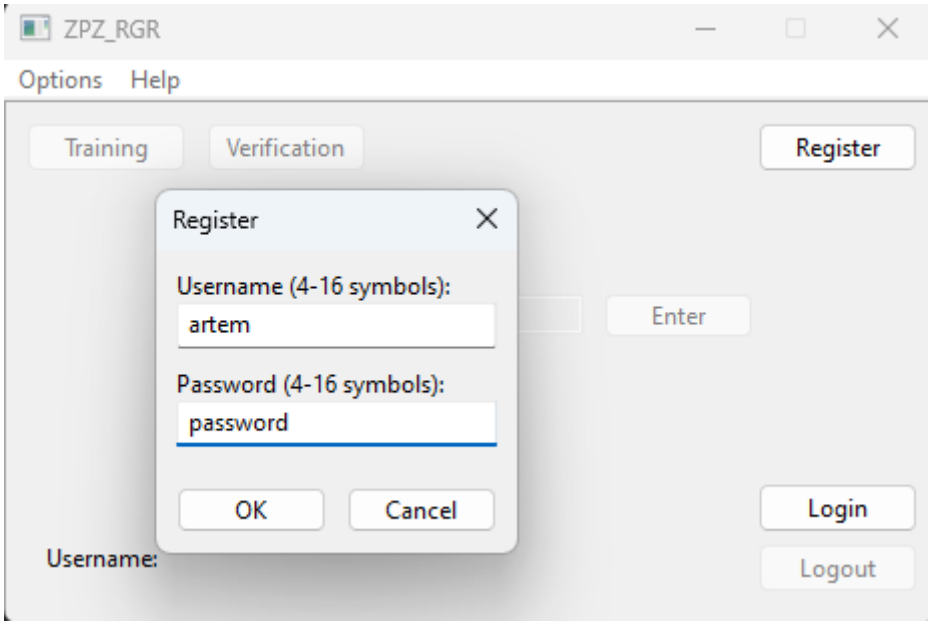
Якщо говорити про коефіцієнт Фішеру, то, для початку, ми відфільтровуємо помилки к еталоні і рахуємо дисперсії еталонної вибірки і поточної. Обирається більша з двох дисперсій і розраховується коефіцієнт Фішера для цих двох вибірок, де більша ділиться на меншу. Далі ми отримуємо табличний коефіцієнт Фішера і порівнюємо з отриманим при перевірці біометрії. Якщо ж отриманий коефіцієнт більший за табличний – людина не допускається до облікового запису.

Якщо ж було пройдено тест з коефіцієнтом Фішера, то далі йде другий тест з коефіцієнтом Стьюдента. Спочатку ми знову обраховуємо матсподівання і дисперсію, і хоч у нас є для цього спеціальний метод, в даному випадку я зробив обчислення в самій функції, бо нам потрібні матсподівання, які нам прийшлось обраховувати два разу, якщо я викликав свій же метод обрахування дисперсій (в самому методі обрахування і в «checkSample»). Далі береться табличний коефіцієнт Стьюдента і якщо він менший за отриманий нами – користувач не допускається. Інакше біометричний тест пройдено успішно.

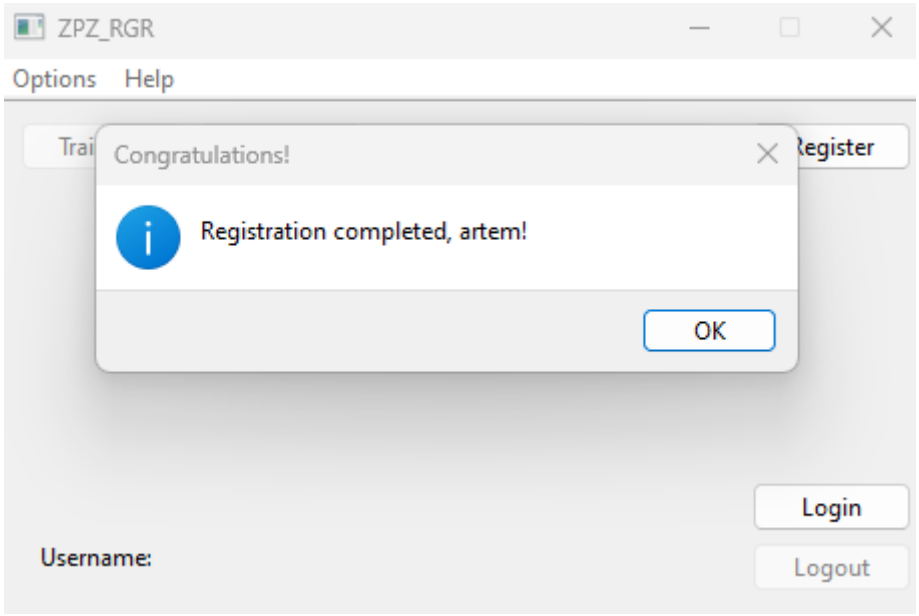
Нижче наведено як працює програма:

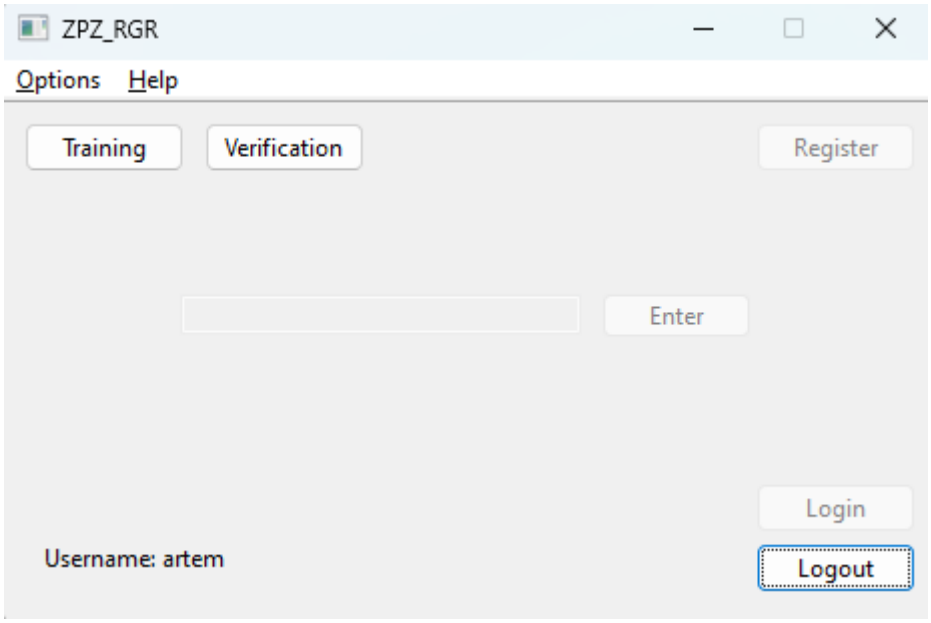


Маємо початковий екран

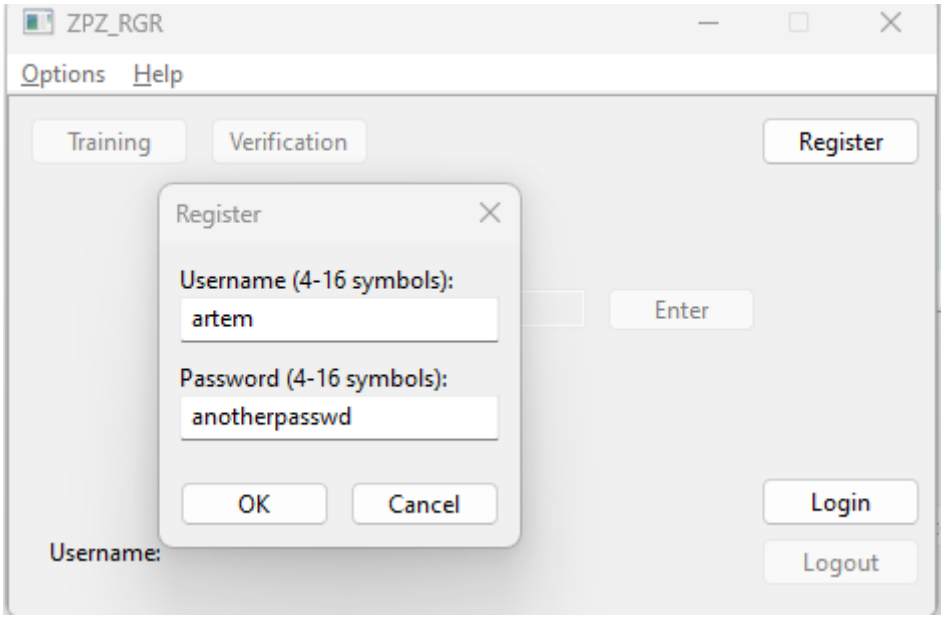


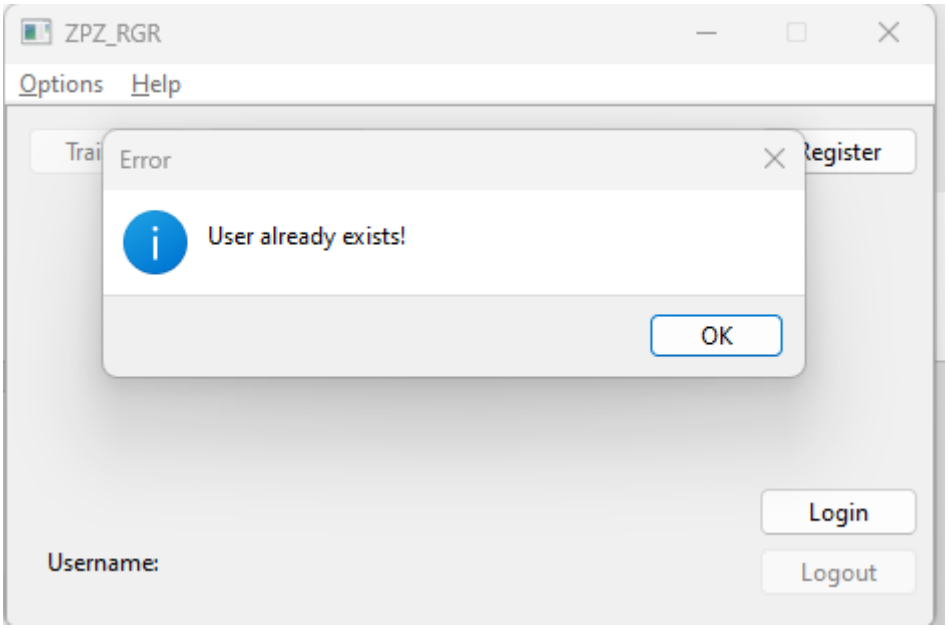
Реєструємо аккаунт



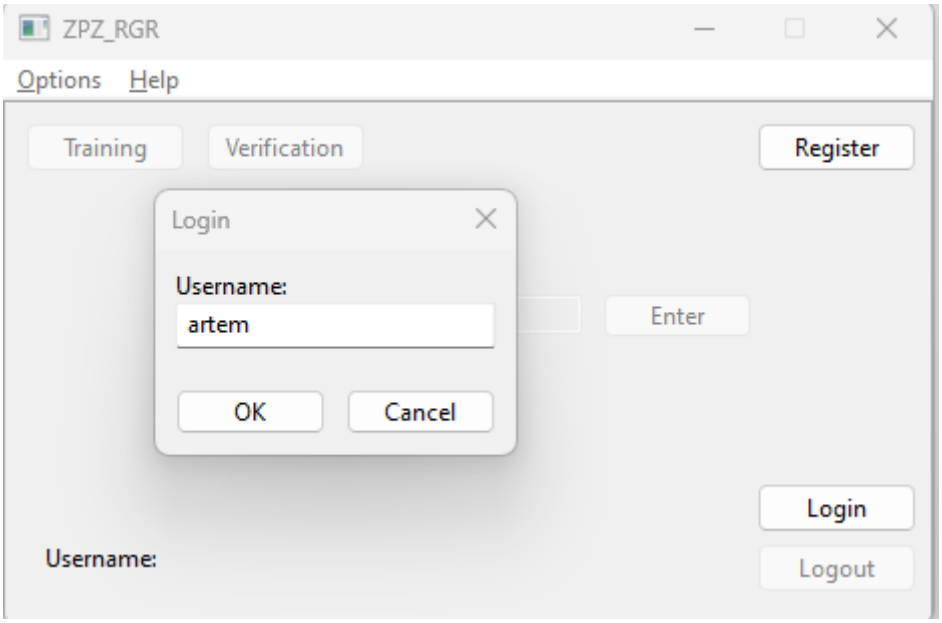


Відповідно успішно реєструємося.

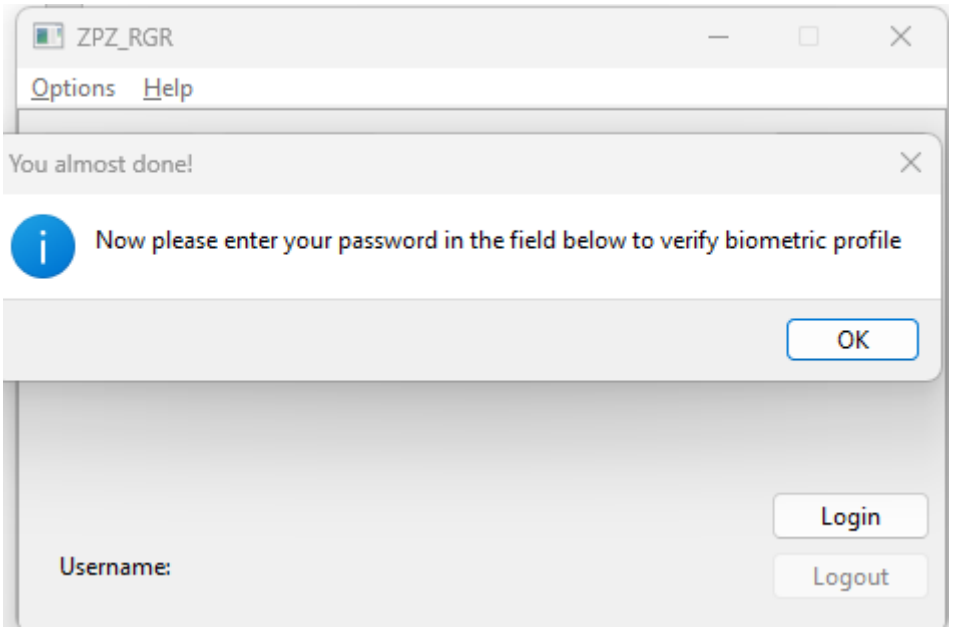




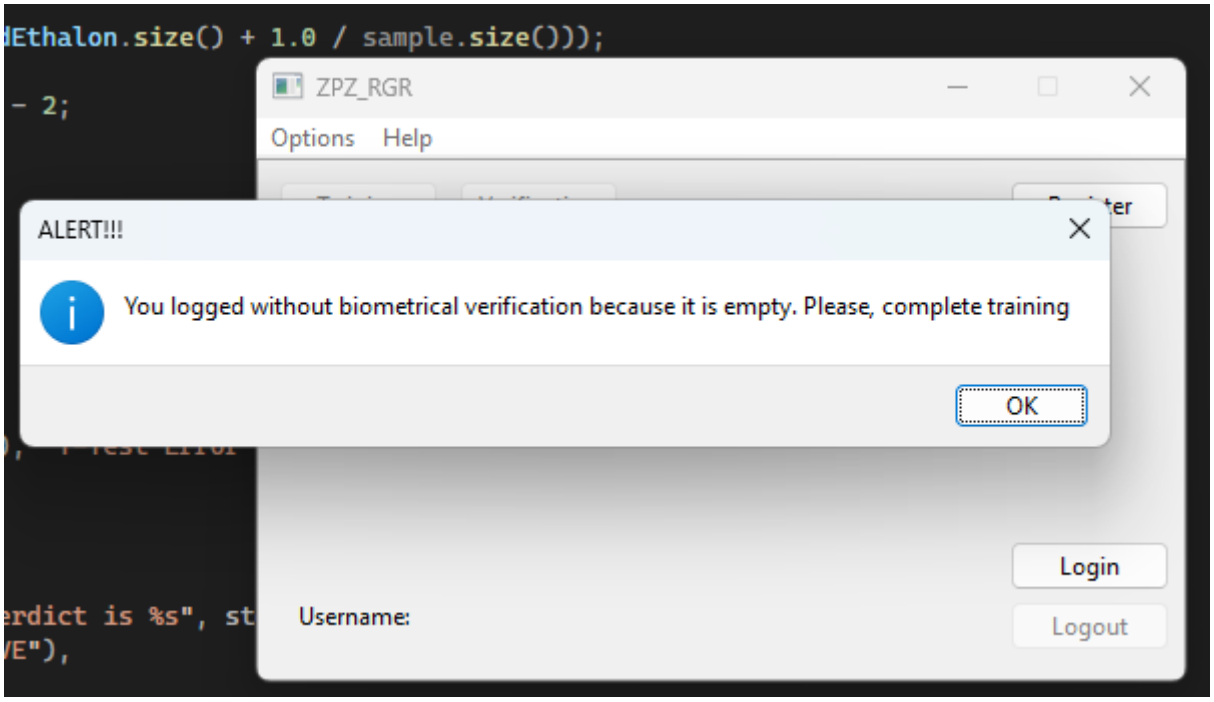
При реєстрації юзеру з таким же ім’ям – помилка реєстрації



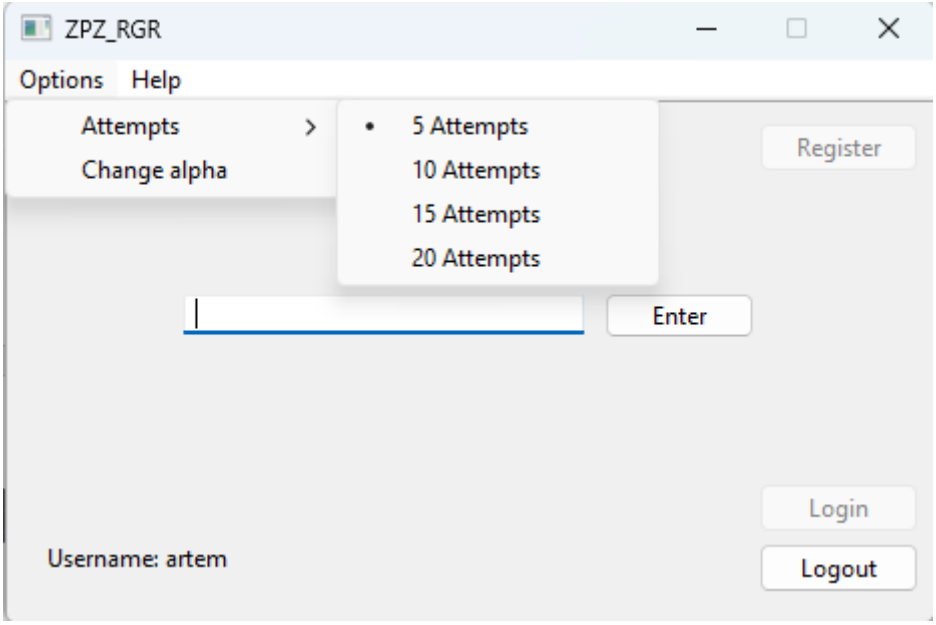
Логінимося

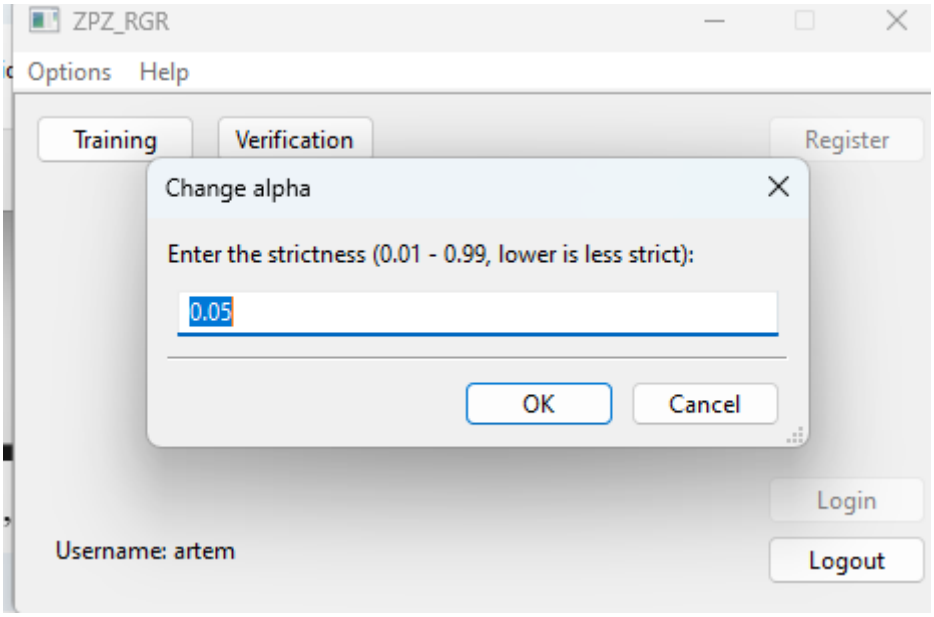


Бачимо прохання бути готовими до проходу біометричного тесту, але наразі його не буде, бо не пройдено тренування.

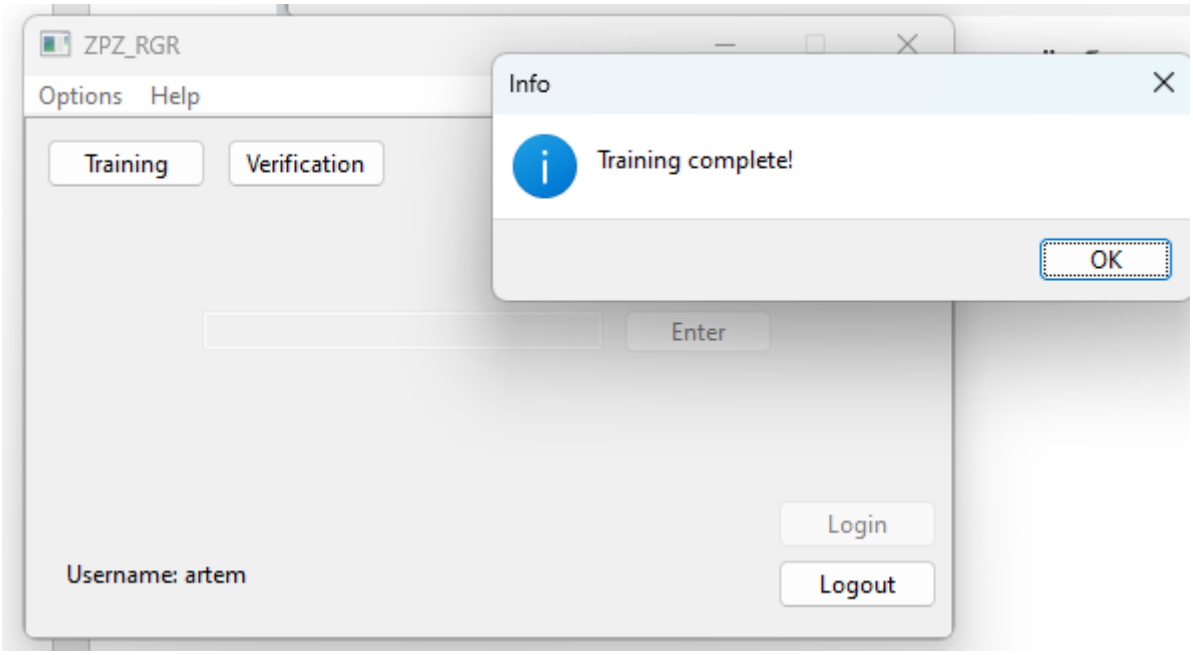


На що і виводиться попередження, що треба натренувати біометричну базу.

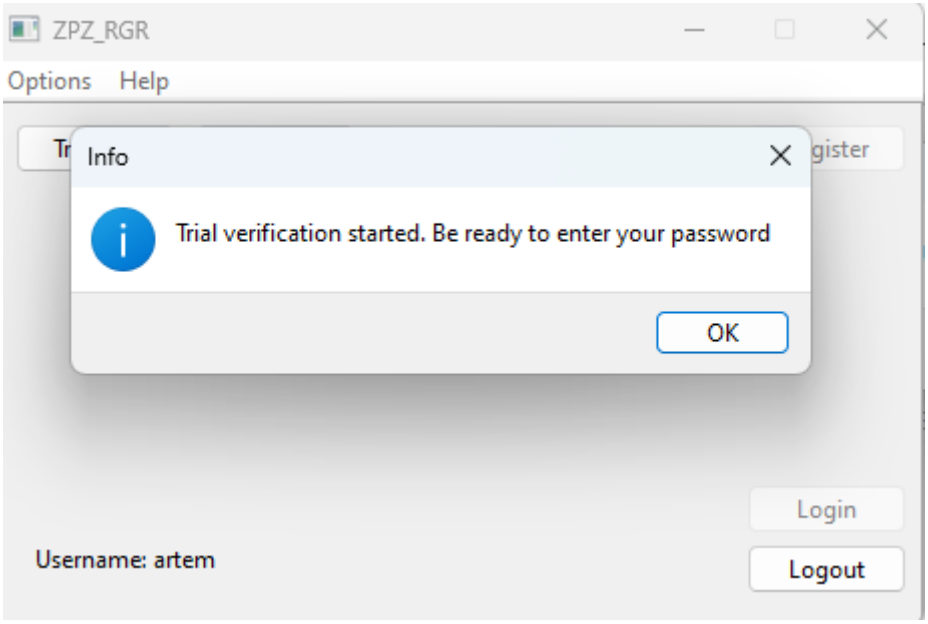




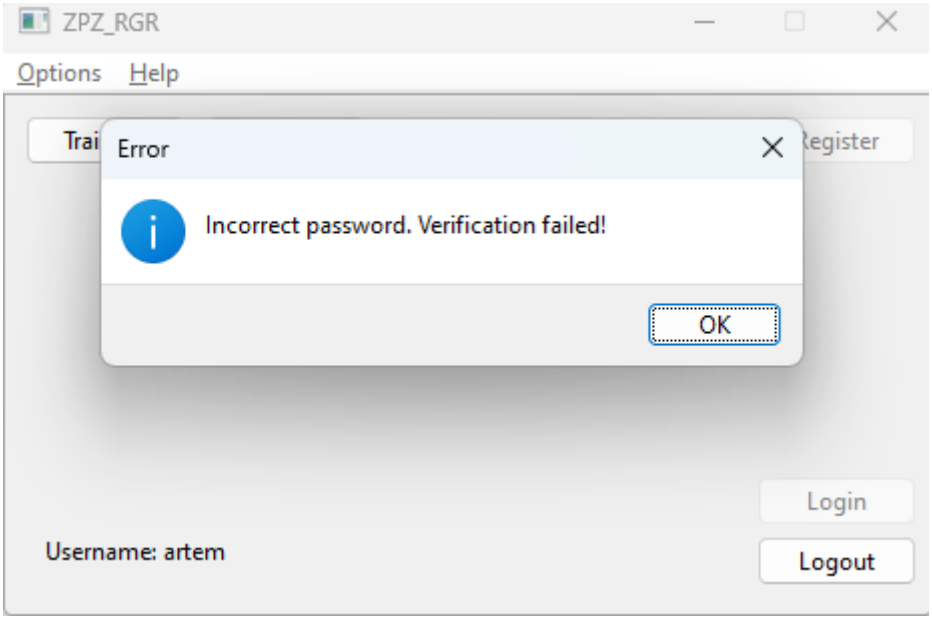
Можна змінити налаштування безпеки, залишу їх базовими.



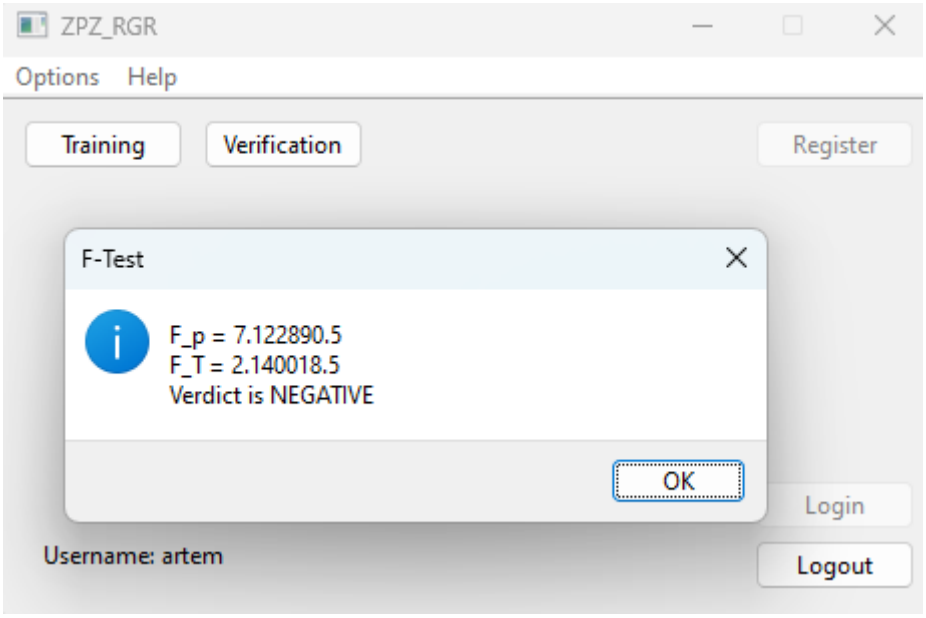
Проходимо тренування.

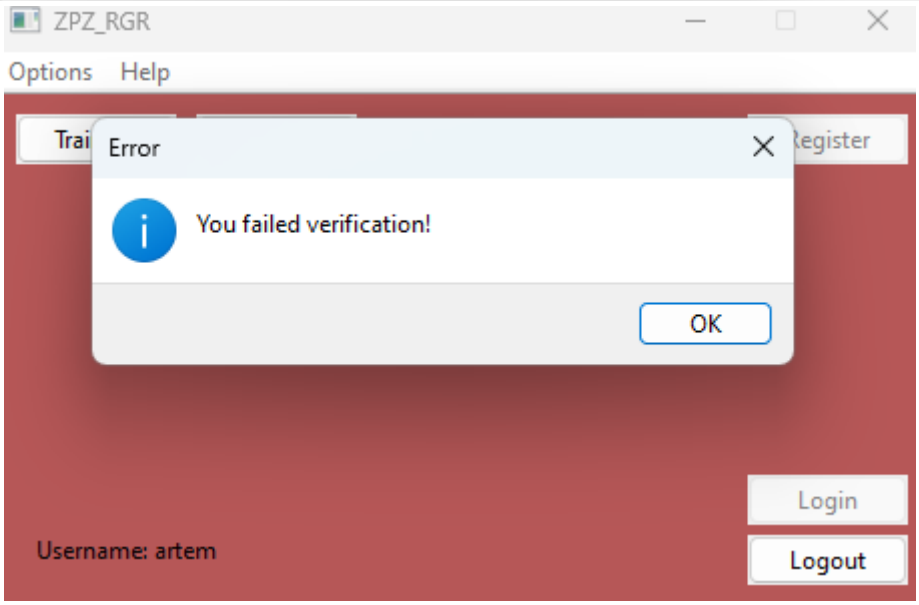


Тепер пройдемо пробну верифікацію.

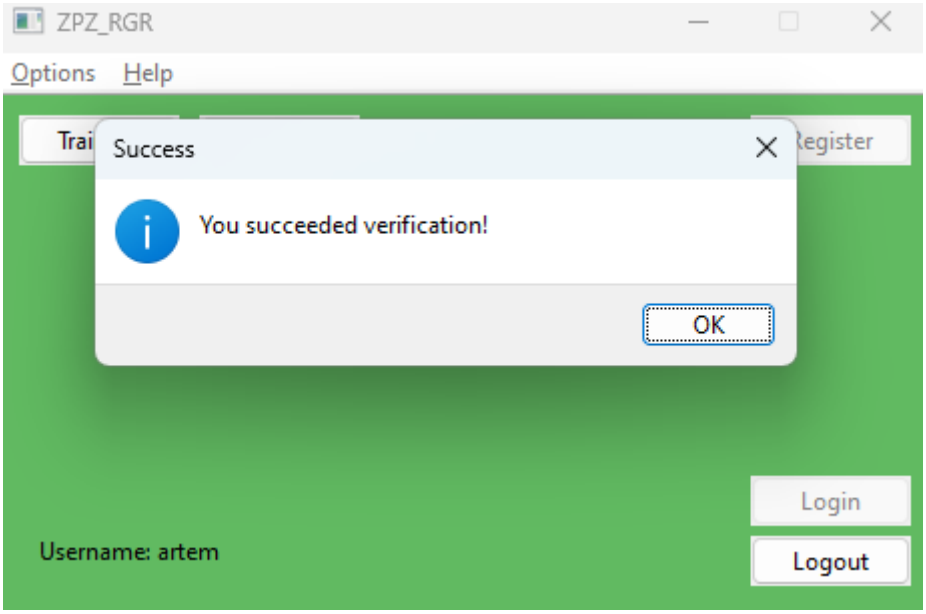


При неправильному вводі паролю – помилка.

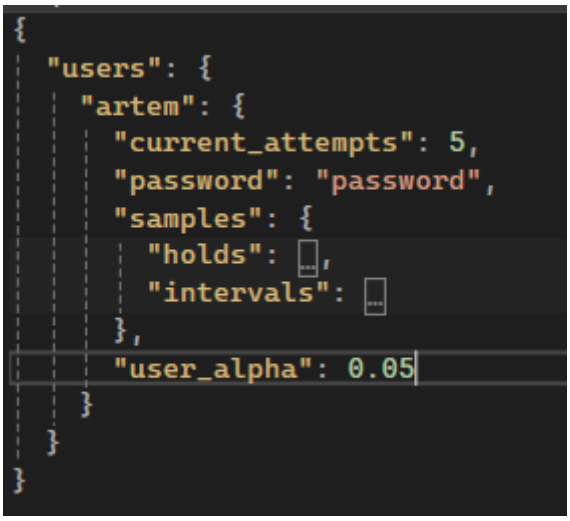




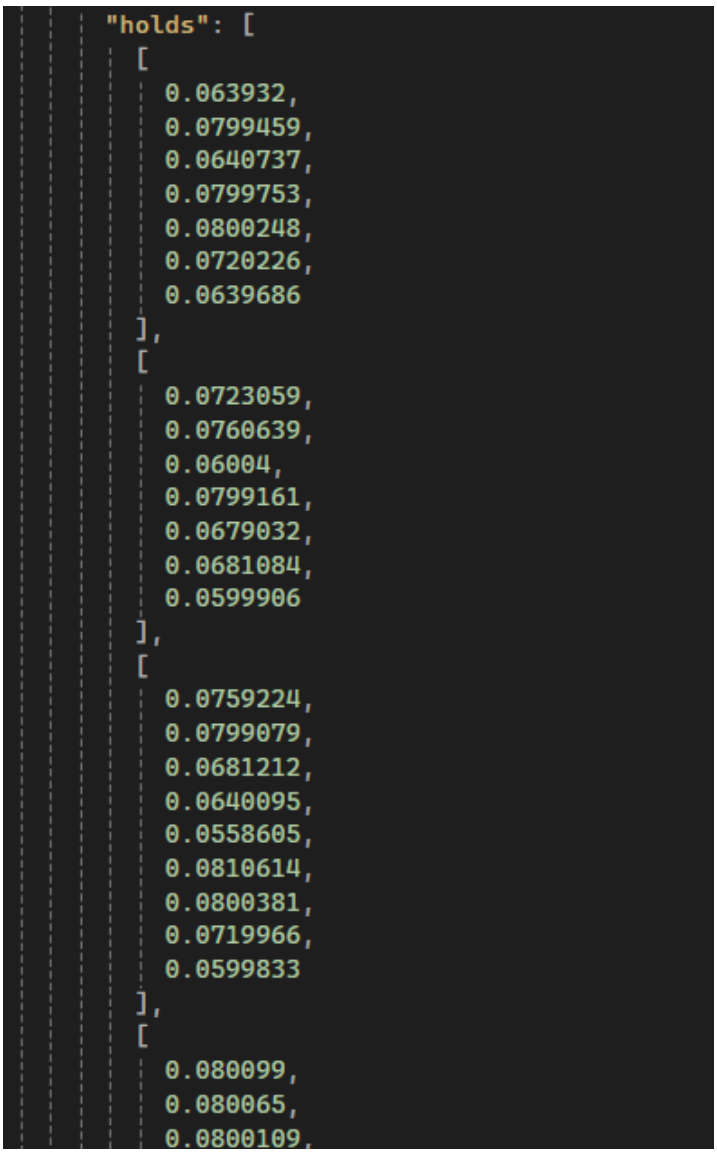
Якщо біометричний «почерк» не той (я спеціально ввів пароль дуже повільно) – помилка. (І так, при помилці біометричної перевірки – фон світиться червоним, а при правильній перевірці - зеленим)



Якщо перевірка пройдено успішно – відповідно і повідомлення про це сповіщає.



Ось так виглядає БД (поки тут один користувач, але сам шаблон видний).



В «holds» і «intervals», відповідно, такі вибірки.

III. РЕЗУЛЬТАТИ ТЕСТІВ НА ПОМИЛКИ ДРУГОГО І ПЕРШОГО РОДУ

Помилки 1-го роду:	Помилки 2-го роду:
success	negative
success	negative
success	negative
success	negative
negative	success
success	negative
negative	negative
success	negative
success	negative
negative	negative
success	negative
success	negative
negative	negative
success	negative
negative	success
success	negative
success	negative
success	negative
negative	negative
success	negative
0,3	0,1

(Успішний вхід – success, неуспішний – negative)

Помилки першого роду розраховував як кількість негативних спроб входу на всі спроби (20 тестів), заходив я сам в свій аккаунт. Для тесту на помилки другого роду я посадив подругу, яка доволі часто працює за комп’ютером, вона адміністратор певного закладу і часто працює з документацією, а тому теж має певний почерк. Ділив кількість успішних спроб заходу на всі.

IV. ВИСНОВКИ

Під час виконання розрахунково-графічної роботи я засвоїв як і нову бібліотеку для роботи з GUI, але це додатково, переш за все я засвоїв методі біометричної верифікації користувачів, як вони працюють в теорії і на практиці. Які для цього використовуються математичні методи теорії ймовірностей і математичної статистики, а саме коефіцієнт Фішеру для порівняння дисперсій поточної спроби з еталонною вибіркою, а також критерій Стьюдента для того, щоб оцінити, наскільки середній час натискання (або інтервалу) у поточній спробі відрізняється від еталонної вибірки, а також як саме вони розраховуються. А також як ці математичні методи можна імплементувати в додаток для того, щоб повисити його стійкість до неавторизованого доступу до облікових записів.