Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

(МТУСИ)

Лабораторная работа №2 по теме:

«МЕТОДЫ ПАРОЛЬНОЙ ЗАЩИТЫ. РАЗРАБОТКА ПРОГРАММНОЙ ПАРОЛЬНОЙ ЗАЩИТЫ»

Вариант №1

Выполнил студент группы БФИ2202

Сидорук Данил

**Цель работы**

Изучение технологии аутентификации пользователя на основе пароля.

**Ход работы**

1. Ознакомиться с теоретической частью данной работы.

2. Составить программу, представляющую собой форму доступа к определённым информационным ресурсам на основе пароля.

3. Составить отчет по проделанной работе.

4. Защитить работу.

**Постановка задачи**

Разработать программу, представляющую собой форму доступа к определённым информационным ресурсам на основе пароля.

1. В качестве информационного ресурса использовать любой файл или приложение.

2. Доступ к ресурсу должен быть разрешен только санкционированным пользователям. Для этого в программе должны храниться имена пользователей и их пароли. При попытке доступа пользователя к ресурсу проверяется наличие его идентификатора (имени) в системе и соответствие введеного пароля паролю, который хранится в системе.

3. В системе должна храниться следующая информация о пользователе: ID или имя пользователя, пароль, ФИО, дата рождения, место рождения (город), номер телефона.

4. Пользователь должен иметь возможность поменять пароль (таблица 2).

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Длина пароля (количество символов) | Используемые символы | Дополнительные методы защиты |
| 1 | 6 | *Латинский алфавит (строчные буквы)* | При смене пароля: проверка на отсутствие повторяющихся символов |

**Листинг программы**

*authorizationform.cpp*

#include "authorizationform.hpp"  
#include "../database\_manager.hpp"  
#include "registrationform.hpp"  
  
#include <QMessageBox>  
  
AuthorizationForm::AuthorizationForm(QWidget \*parent) : QMainWindow(parent) {  
 ui.setupUi(this);  
  
 connect(ui.login\_button, &QPushButton::clicked, this, [this](bool) {  
 const auto &username = ui.username\_edit->text(),  
 &pass = ui.password\_edit->text();  
  
 if (username.isEmpty() || pass.isEmpty()) {  
 QMessageBox::critical(this, "Вход невозможен",  
 "Введите все необходимые данные");  
 return;  
 }  
  
 DatabaseManager \*databaseManager =  
 **qApp**->property("databaseManager").value<DatabaseManager \*>();  
  
 if (!databaseManager->login(username, pass)) {  
 QMessageBox::critical(this, "Вход невозможен", "Неверные данные");  
 return;  
 }  
  
 QMessageBox::information(this, "Успех",  
 "Поздравляю, вы успешно зашли в аккаунт");  
 **qApp**->setProperty("username", QVariant::fromValue(username));  
 **qApp**->setProperty("pass", QVariant::fromValue(pass));  
 this->close();  
 });  
  
 connect(ui.signup\_button, &QPushButton::clicked, this, [this](bool) {  
 this->close();  
 RegistrationForm \*registrationForm = new RegistrationForm(this);  
 registrationForm->show();  
 });  
}

*authorizationform.hpp*

#include "ui\_authorization.h"  
  
class AuthorizationForm : public QMainWindow {  
 **Q\_OBJECT**public:  
 explicit AuthorizationForm(QWidget \*parent = nullptr);  
  
private:  
 Ui::Authorization ui;  
};

*changeform.cpp*

#include "changeform.hpp"  
#include "../database\_manager.hpp"  
#include "../validators.h"  
  
#include <QMessageBox>  
  
ChangeForm::ChangeForm(QWidget \*parent) : QMainWindow(parent) {  
 ui.setupUi(this);  
  
 connect(ui.save\_button, &QPushButton::clicked, this, [this](bool) {  
 const auto &current\_pass = ui.current\_password\_edit->text(),  
 &new\_pass = ui.new\_password\_edit->text(),  
 &control\_pass = ui.control\_password\_edit->text();  
  
 if (current\_pass.isEmpty() || new\_pass.isEmpty() ||  
 control\_pass.isEmpty()) {  
 QMessageBox::critical(this, "Ошибка", "Введите все необходимые данные");  
 return;  
 }  
  
 if (new\_pass != control\_pass) {  
 QMessageBox::critical(this, "Ошибка",  
 "Новый пароль не совпадает с контрольным паролем");  
 return;  
 }  
  
 if (current\_pass == new\_pass) {  
 QMessageBox::critical(this, "Ошибка",  
 "Новый пароль должен быть отличен от текущего");  
 return;  
 }  
  
 PassValidator passValidator(new\_pass);  
 if (!passValidator.validate()) {  
 QMessageBox::critical(this, "Сохранение невозможно",  
 passValidator.getErrorMessage());  
 return;  
 }  
  
 DatabaseManager \*databaseManager =  
 **qApp**->property("databaseManager").value<DatabaseManager \*>();  
 const QString &username = **qApp**->property("username").value<QString>();  
  
 if (!databaseManager->changePass(username, current\_pass, new\_pass)) {  
 QMessageBox::critical(this, "Ошибка", "Неверный пароль");  
 return;  
 }  
  
 QMessageBox::information(this, "Успех",  
 "Поздравляю, вы успешно сменили пароль");  
 **qApp**->setProperty("username", QVariant::fromValue(username));  
 **qApp**->setProperty("pass", QVariant::fromValue(new\_pass));  
 this->close();  
 });  
}

*changeform.hpp*

#include "ui\_change.h"  
  
class ChangeForm : public QMainWindow {  
 **Q\_OBJECT**public:  
 explicit ChangeForm(QWidget \*parent = nullptr);  
  
private:  
 Ui::Change ui;  
};

*mainwindowform.cpp*

#include "mainwindowform.hpp"  
  
#include "../database\_manager.hpp"  
#include "authorizationform.hpp"  
#include "changeform.hpp"  
#include "registrationform.hpp"  
  
#include <QMessageBox>  
#include <qmessagebox.h>  
  
MainWindowForm::MainWindowForm(QWidget \*parent) : QMainWindow(parent) {  
 ui.setupUi(this);  
  
 connect(ui.login\_button, &QPushButton::clicked, this, [this](bool) {  
 AuthorizationForm \*authorizationForm = new AuthorizationForm(this);  
 authorizationForm->show();  
 });  
  
 connect(ui.signup\_button, &QPushButton::clicked, this, [this](bool) {  
 RegistrationForm \*registrationForm = new RegistrationForm(this);  
 registrationForm->show();  
 });  
  
 connect(ui.change\_button, &QPushButton::clicked, this, [this](bool) {  
 DatabaseManager \*databaseManager =  
 **qApp**->property("databaseManager").value<DatabaseManager \*>();  
 const QString &username = **qApp**->property("username").value<QString>(),  
 &pass = **qApp**->property("pass").value<QString>();  
  
 if (!databaseManager->login(username, pass)) {  
 QMessageBox::critical(this, "Ошибка", "Вы не авторизованы");  
 return;  
 }  
  
 ChangeForm \*changeForm = new ChangeForm(this);  
 changeForm->show();  
 });  
  
 connect(ui.access\_button, &QPushButton::clicked, this, [this](bool) {  
 DatabaseManager \*databaseManager =  
 **qApp**->property("databaseManager").value<DatabaseManager \*>();  
  
 const QString &username = **qApp**->property("username").value<QString>(),  
 &pass = **qApp**->property("pass").value<QString>();  
  
 if (!databaseManager->login(username, pass)) {  
 QMessageBox::critical(this, "Ошибка", "Вы не авторизованы");  
 return;  
 }  
  
 QMessageBox::information(this, "Успех", "Узрите секретные данные: 42");  
 });  
}

*mainwindowform.hpp*

#include "ui\_mainwindow.h"  
  
class MainWindowForm : public QMainWindow {  
 **Q\_OBJECT**public:  
 explicit MainWindowForm(QWidget \*parent = nullptr);  
  
private:  
 Ui::MainWindow ui;  
};

*registrationform.cpp*

#include "registrationform.hpp"  
#include "../database\_manager.hpp"  
#include "../validators.h"  
  
#include <QMessageBox>  
  
RegistrationForm::RegistrationForm(QWidget \*parent) : QMainWindow(parent) {  
 ui.setupUi(this);  
  
 connect(ui.save\_button, &QPushButton::clicked, this, [this](bool) {  
 const auto &username = ui.username\_edit->text(),  
 &pass = ui.password\_edit->text(),  
 &lastName = ui.last\_name\_edit->text(),  
 &firstName = ui.name\_edit->text(),  
 &patronymic = ui.patronymic\_edit->text(),  
 &birthPlace = ui.birthplace\_edit->text(),  
 &phone = ui.phone\_edit->text();  
 const auto &birthDate = ui.birthdate\_edit->date();  
  
 if (username.isEmpty() || pass.isEmpty() || lastName.isEmpty() ||  
 firstName.isEmpty() || patronymic.isEmpty() || birthDate.isNull() ||  
 birthPlace.isEmpty() || phone.isEmpty()) {  
 QMessageBox::critical(this, "Сохранение невозможно",  
 "Введите все необходимые данные");  
 return;  
 }  
  
 PassValidator passValidator(pass);  
 if (!passValidator.validate()) {  
 QMessageBox::critical(this, "Сохранение невозможно",  
 passValidator.getErrorMessage());  
 return;  
 }  
  
 DatabaseManager \*databaseManager =  
 **qApp**->property("databaseManager").value<DatabaseManager \*>();  
  
 if (!databaseManager->createUser(username, pass, lastName, firstName,  
 patronymic, birthDate, birthPlace,  
 phone)) {  
 QMessageBox::critical(this, "Сохранение невозможно",  
 "Вероятно, username уже занят");  
 return;  
 }  
  
 QMessageBox::information(this, "Успех",  
 "Поздравляю, вы успешно зарегистрировались");  
 this->close();  
 });  
}

*registrationform.hpp*

#include "ui\_registration.h"  
  
class RegistrationForm : public QMainWindow {  
 **Q\_OBJECT**public:  
 explicit RegistrationForm(QWidget \*parent = nullptr);  
  
private:  
 Ui::Registration ui;  
};

*database\_manager.cpp*

#include "database\_manager.hpp"  
  
#include <QDebug>  
#include <QSqlError>  
#include <QSqlQuery>  
#include <qurl.h>  
  
DatabaseManager::DatabaseManager(const QString &path) {  
 database = QSqlDatabase::addDatabase("QSQLITE");  
 database.setDatabaseName(path);  
  
 if (!database.open()) {  
 **qDebug**() << "Error: connection with database";  
 } else {  
 **qDebug**() << "Database: connection ok";  
 }  
}  
  
bool DatabaseManager::createUser(  
 const QString &username, const QString &pass, const QString &lastName,  
 const QString &firstName, const QString &patronymic, const QDate &birthDate,  
 const QString &birthPlace, const QString &phone) {  
  
 QSqlQuery selectQ;  
 selectQ.prepare(  
 R"(SELECT "username" FROM "main"."users" WHERE username = ?)");  
 selectQ.addBindValue(username);  
 selectQ.exec();  
  
 if (!selectQ.exec()) {  
 **qDebug**() << "selectUser error:" << selectQ.lastError();  
 return false;  
 }  
  
 **qDebug**() << "selectUser status:" << "ok";  
 if (selectQ.next()) {  
 **qDebug**() << "selectUser collision";  
 return false;  
 }  
  
 QSqlQuery insertQ;  
 insertQ.prepare(  
 R"(INSERT INTO "main"."users" ("username", "password", "last\_name", "first\_name", "patronymic", "birthdate", "birthplace", "phone") VALUES (?, ?, ?, ?, ?, ?, ?, ?);)");  
  
 insertQ.addBindValue(username);  
 insertQ.addBindValue(pass);  
 insertQ.addBindValue(lastName);  
 insertQ.addBindValue(firstName);  
 insertQ.addBindValue(patronymic);  
 insertQ.addBindValue(birthDate.toString());  
 insertQ.addBindValue(birthPlace);  
 insertQ.addBindValue(phone);  
  
 if (!insertQ.exec()) {  
 **qDebug**() << "createUser error:" << insertQ.lastError();  
 return false;  
 }  
  
 return true;  
}  
  
bool DatabaseManager::changePass(const QString &username, const QString &pass,  
 const QString &controlPass) {  
 if (!login(username, pass)) {  
 return false;  
 }  
  
 QSqlQuery updateQ;  
 updateQ.prepare(  
 R"(UPDATE "main"."users" SET password = ? WHERE username = ?)");  
 updateQ.addBindValue(controlPass);  
 updateQ.addBindValue(username);  
  
 if (!updateQ.exec()) {  
 **qDebug**() << "updateUser error:" << updateQ.lastError();  
 return false;  
 }  
 **qDebug**() << "updateUser status:" << "ok";  
  
 return true;  
}  
  
bool DatabaseManager::login(const QString &username, const QString &pass) {  
 QSqlQuery selectQ;  
 selectQ.prepare(  
 R"(SELECT 1 FROM "main"."users" WHERE username = ? AND password = ?)");  
 selectQ.addBindValue(username);  
 selectQ.addBindValue(pass);  
  
 if (!selectQ.exec()) {  
 **qDebug**() << "selectUser error:" << selectQ.lastError();  
 return false;  
 }  
 if (!selectQ.next()) {  
 return false;  
 }  
 **qDebug**() << "selectUser status:" << "ok";  
  
 return true;  
}

*database\_manager.hpp*

#include <QDate>  
#include <QSqlDatabase>  
#include <QString>  
  
class DatabaseManager {  
public:  
 DatabaseManager(const QString &path);  
  
 bool createUser(const QString &username, const QString &pass,  
 const QString &lastName, const QString &firstName,  
 const QString &patronymic, const QDate &birthDate,  
 const QString &birthPlace, const QString &phone);  
  
 bool changePass(const QString &username, const QString &pass,  
 const QString &controlPass);  
  
 bool login(const QString &username, const QString &pass);  
  
private:  
 QSqlDatabase database;  
};

*main.cpp*

#include <QDir>  
#include <QFile>  
#include <QFileDevice>  
#include <QStandardPaths>  
#include <QVariant>  
#include <memory>  
  
#include "forms/mainwindowform.hpp"  
  
#include "database\_manager.hpp"  
  
int main(int argc, char \*argv[]) {  
 QApplication app(argc, argv);  
 app.setOrganizationName("eoanermine");  
 app.setApplicationName("pass\_protection");  
  
 QFile resourcesDatabase{":/pass\_protection.database"};  
 if (!resourcesDatabase.open(QIODevice::*ReadOnly*)) {  
 **qFatal**("Can't open resources database");  
 }  
 std::shared\_ptr<DatabaseManager> databaseManager = nullptr;  
  
 auto path = QStandardPaths::writableLocation(QStandardPaths::*AppDataLocation*);  
 if (path.isEmpty())  
 **qFatal**("Cannot determine settings storage location");  
 QDir dir{path};  
 if (dir.mkpath(dir.absolutePath()) && QDir::setCurrent(dir.absolutePath())) {  
 **qDebug**() << "database in" << QDir::currentPath();  
 auto databasePath =  
 dir.filesystemAbsolutePath() / "pass\_protection.database";  
 resourcesDatabase.copy(databasePath);  
 QFile database{databasePath};  
 database.setPermissions(QFileDevice::*WriteOwner* | QFileDevice::*ReadOwner*);  
 databaseManager =  
 std::make\_shared<DatabaseManager>("pass\_protection.database");  
 app.setProperty("databaseManager", QVariant::fromValue(databaseManager));  
 }  
  
 app.setProperty("username", QVariant::fromValue(QString()));  
 app.setProperty("pass", QVariant::fromValue(QString()));  
  
 MainWindowForm mainwindow;  
 mainwindow.show();  
 return app.exec();  
}

*validators.cpp*

#include "validators.h"  
  
PassValidator::PassValidator(const QString &pass): pass(pass) { }  
  
bool PassValidator::validate() {  
 if (pass.length() < 6) {  
 errorMessage = "Пароль должен содержать шесть или больше символов";  
 return false;  
 }  
  
 bool lower\_latin = true;  
 for (QChar x: pass) {  
 if (!(x >= 'a' && x <= 'z')) {  
 lower\_latin = false;  
 break;  
 }  
 }  
  
 if (!lower\_latin) {  
 errorMessage = "Пароль должен состоять исключительно из строчных латинских букв";  
 return false;  
 }  
  
 return true;  
}  
  
const QString& PassValidator::getErrorMessage() {  
 return errorMessage;  
}

*vaidators.h*

#ifndef **PASS\_PROTECTION\_VALIDATORS\_H**#define **PASS\_PROTECTION\_VALIDATORS\_H**#include <QString>  
  
class PassValidator {  
public:  
 PassValidator(const QString& pass);  
 bool validate();  
 const QString& getErrorMessage();  
private:  
 const QString& pass;  
 QString errorMessage;  
};  
  
#endif *//PASS\_PROTECTION\_VALIDATORS\_H*

**Результат выполнения программы**

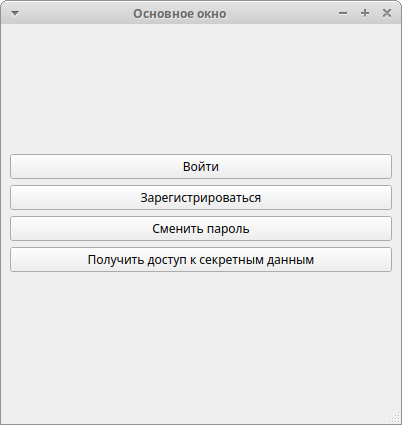


Рисунок 1 - основное окно программы

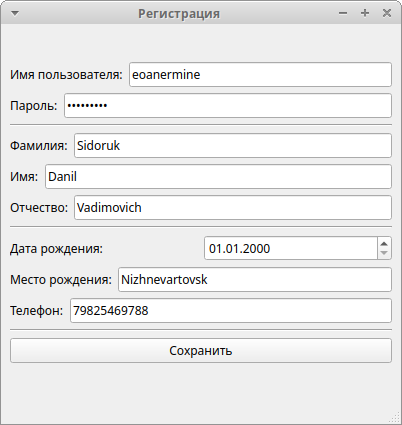


Рисунок 2 – окно регистрации

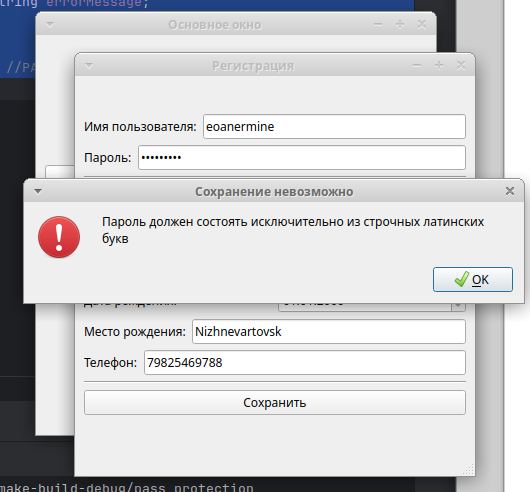


Рисунок 3 - ошибка при попытке ввести неудовлетворяющий требованиям пароль

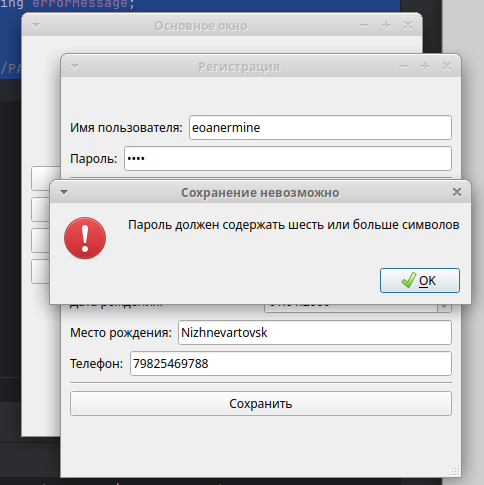


Рисунок 4 - ошибка при попытке ввести неудовлетворяющий требованиям пароль

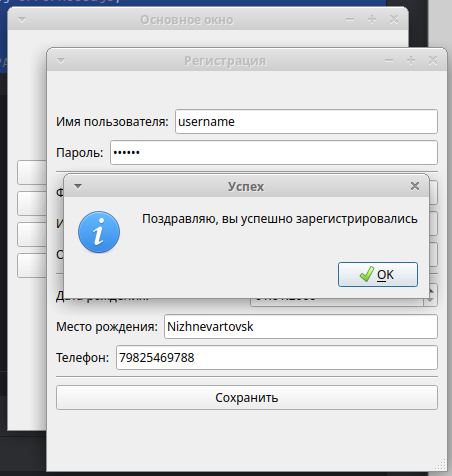


Рисунок 5 – сообщение об успешной регистрации

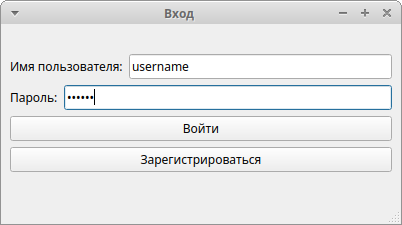


Рисунок 6 – окно авторизации

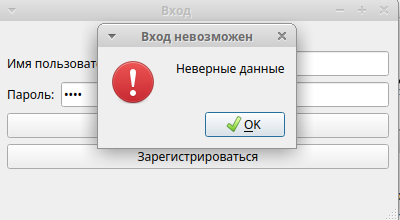


Рисунок 6 – сообщение о неуспешной авторизации

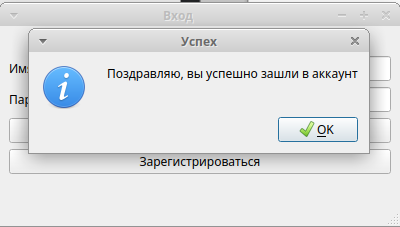


Рисунок 7 – сообщение об успешной авторизации

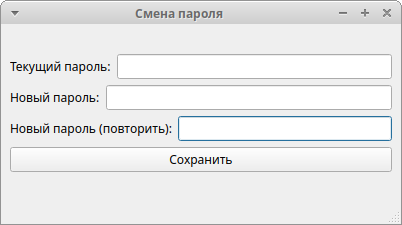


Рисунок 8 – окно смены пароля

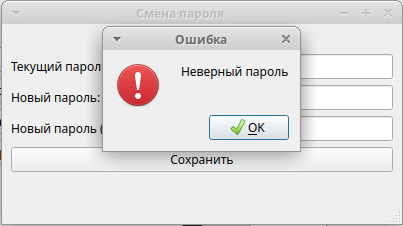


Рисунок 9 – сообщение о неверно введеном пароле

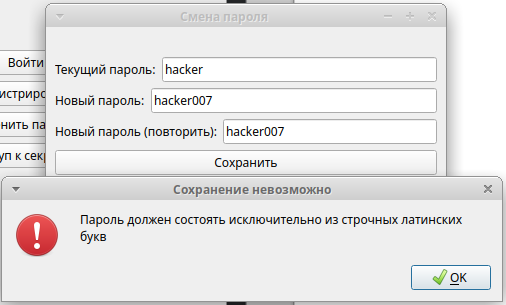


Рисунок 10 – сообщение о неудовлетворяющем условиям пароле

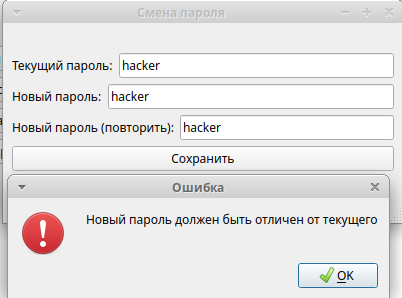


Рисунок 11 – сообщение о неудовлетворяющем условиям пароле

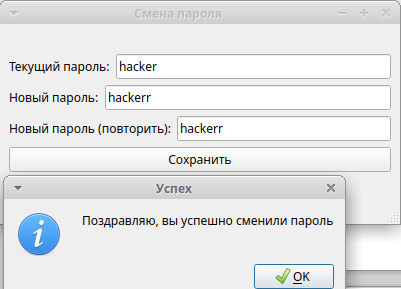


Рисунок 12 – сообщение об успешной смене пароля

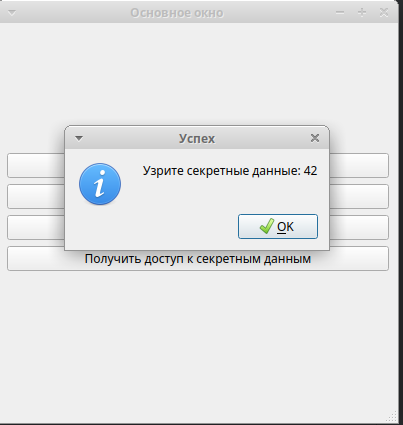


Рисунок 13 – доступ к секретным данным для авторизованных лиц

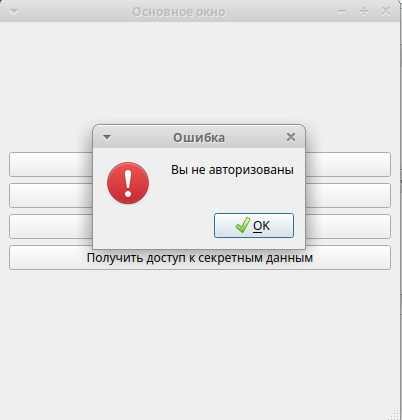


Рисунок 14 – доступ к секретным данным для неавторизованных лиц