Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий

Кафедра «Информационные системы»

Дисциплина «Информационная безопасность»

Лабораторная работа №1

Выполнил:

студент гр. ПИбд – 41

Малафеев Л.С.

Проверил:  
Мытарев П.В.

Ульяновск, 2025г.

**Описание задания**

Отчёт по выполненной лабораторной работе по теме: Разработка программы разграничения полномочий пользователей на основе парольной аутентификации c использованием встроенных криптопровайдеров.

Вариант: 19. Ограничения на выбираемые пароли: Наличие букв и цифр. Криптографические методы. Режим шифрования алгоритма DES – CFB. Есть добавление к ключу случайного значения (Да). Алгоритм хеширования пароля: MD4.

**Скрин-шоты приложения с описанием**

При запуске программы вылезает формочка для ввода пароля расшифровки/зашифровки файла. В ней есть текстовое поле для ввода пароля, кнопка «ОК», кнопка «Отмена».

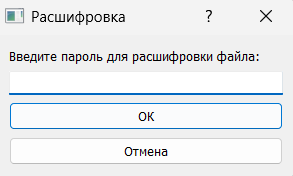


Figure . Форма для ввода пароля базы данных.

Если пароль введён неверно, вылезает ошибка, после которой программа завершает работу:

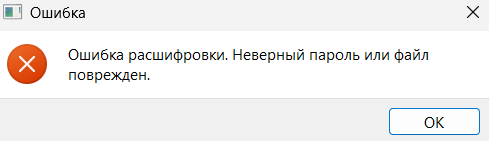


Figure . Ошибка пароля базы данных

Если пароль введён верно, переходим на главную форму. На главной форме имеются три кнопки, две из которых закреплены в меню. Есть кнопка выход – выходит из программы, запрашивая пароль для шифрования базы данных. Справка -> О программе – показывает информацию о программе, авторе и методах. Кнопка войти – открывает формочку входа в кабинет.

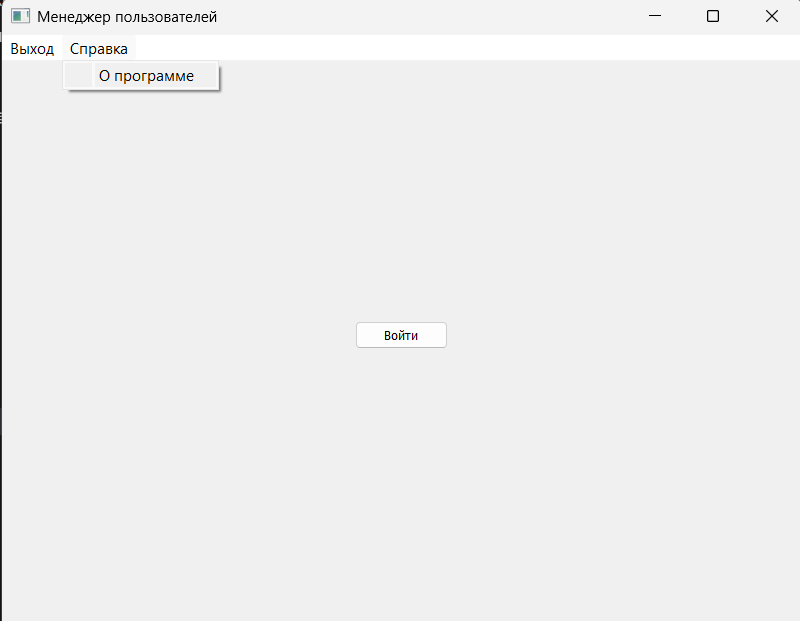


Figure . Главная форма

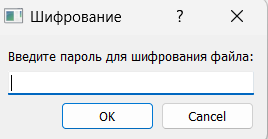


Figure . Форма шифрования файла бд.

При нажатии кнопки войти, открывается форма входа в кабинет. Можно ввести логин и пароль, войти или выйти в главное меню. По старту программы у нас есть пользователь только «ADMIN» с пустым паролем.

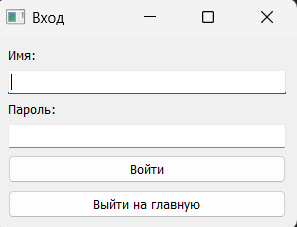


Figure . Форма входа

При входе в панель администратора мы видим:

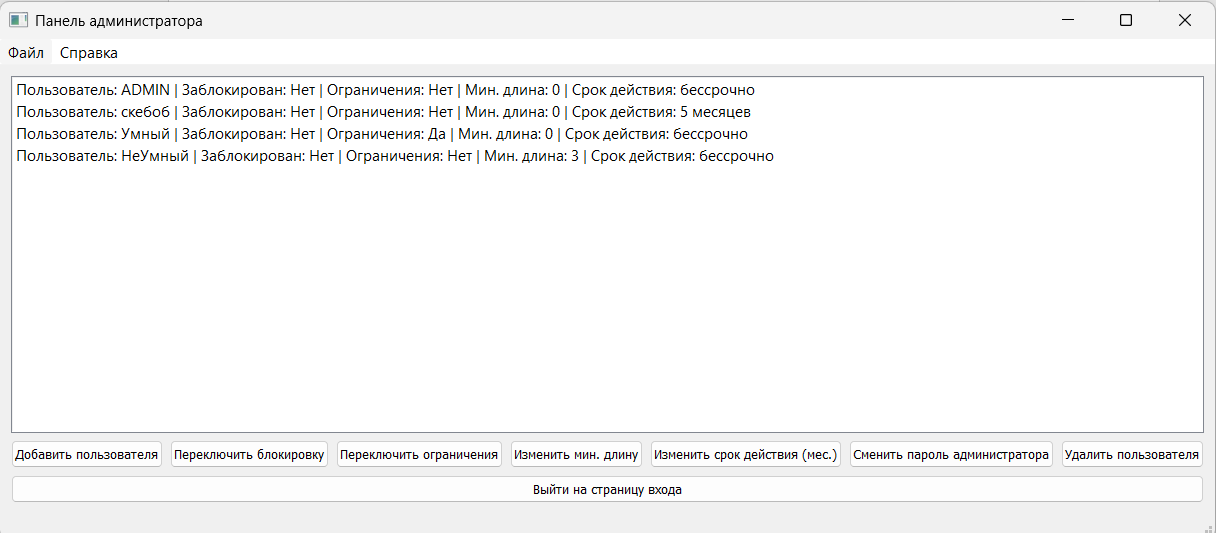


Figure . Панель администратора

В панели администратора мы видим список всех пользователей. Мы можем добавить нового пользователя, указав ему имя.

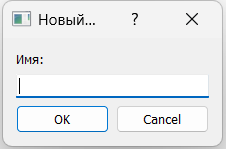


Figure . Добавление пользователя

Кнопка переключить блокировку блокирует или разблокирует пользователя. Если заблокированный пользователь пытается войти в аккаунт, то вылезает ошибка входа. Также можно для пользователя включить ограничения пароля по выданному заданию.

При изменении минимальной длинны пользователю показывается текущая длина и можно изменить на новую.

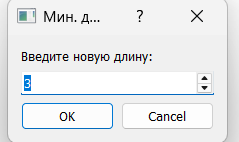


Figure . Изменение длины

При изменении срока действия также указывается числовое значение в месяцах.

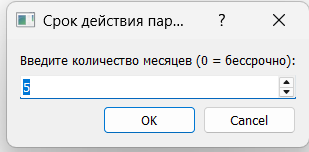


Figure . Измененеие срока

При смене пароля для администратора нужно ввести текущий пароль и дважды новый пароль. То есть подтвердить его.

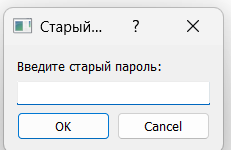


Figure . Ввод старого пароля

Если пароли не совпадают, выходит ошибка:

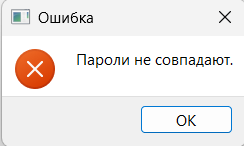


Figure . Ошибка проверки двух паролей.

Если у пользователя стоит ограничение и пароль не подходит под эти ограничения, также вылезает ошибка:

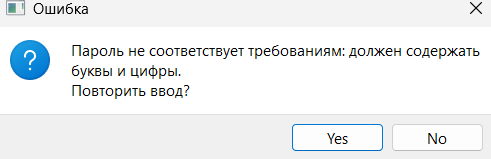


Figure . Не соответствует требованиям

Если также не совпадает минимальная длина – вылезает такое же окно.

Администратор может удалить пользователя. Есть выход на форму входа – после этого сохраняется и происходит выход на главную.

Далее описание процесса входа в пользовательское окно.

В момент, когда пользователь вводит неправильный пароль, вылезает ошибка предупреждение:

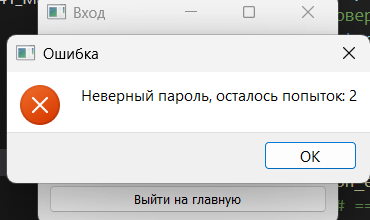


Figure . Неверный пароль.

После ввода трёх попыток – программа завершает работу.

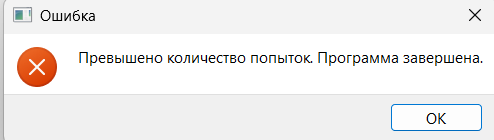


Figure . Завершение работы

Попав в форму пользователя, видим такие кнопки как:

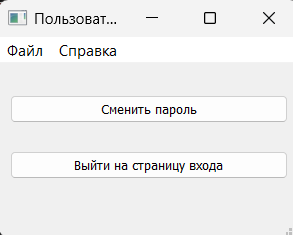


Figure . Форма пользователя

Пароль у пользователя меняется также, как и у администратора. Ввод старого пароля, ввод нового, повторный ввод нового пароля. Если есть ограничения – проверка.

Справка «о программе» выдаёт это:

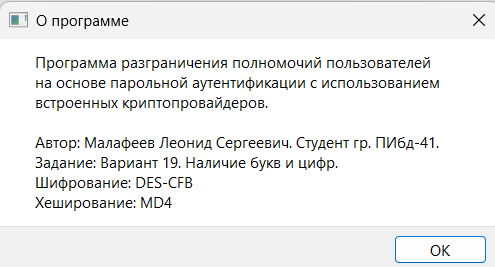


Figure . О программе

**Описание структуры базы данных с пользовательской информацией**

База данных реализована с помощью SQLite. Хранится в зашифрованном виде в файле users.db.enc. При запуске программы файл расшифровывается и временно сохраняется как users.db.tmp, с которым работает приложение. После завершения работы база данных снова шифруется и сохраняется в users.db.enc, а временный файл удаляется.

Таблица Users содержит следующие поля:

1. username — текст, первичный ключ, уникальное имя пользователя.
2. password\_hash — текст, хеш пароля (MD4).
3. locked — целое число (0 или 1), статус блокировки (0 — не заблокирован, 1 — заблокирован).
4. password\_policy — целое число (0 или 1), включены ли ограничения (0 — нет, 1 — да).
5. min\_length — целое число, минимальная длина пароля (0 — без ограничений).
6. valid\_months — целое число, срок действия пароля в месяцах (0 — бессрочно).

**Описание функций, связанных с хешированием и шифрованием**

Хеширование паролей осуществляется с помощью алгоритма MD4. При сохранении пароля в базе данных он не хранится в открытом виде, а преобразуется в хеш. При аутентификации введённый пароль также хешируется и сравнивается с сохранённым хешем.

Используется функция MD4.new() из библиотеки Crypto.Hash, которая создаёт хеш-объект. В методах authenticate и change\_password пароли хешируются перед сравнением.

Шифрование файла базы данных выполняется с помощью алгоритма DES в режиме CFB (Cipher Feedback). Ключ шифрования генерируется из пароля с помощью хеширования MD4, к которому добавляется случайное значение (соль). При шифровании также используется случайный IV (инициализационный вектор), который сохраняется в начале зашифрованных данных.

Функции encrypt\_file и decrypt\_file в файле crypto.py отвечают за шифрование и расшифровку. Они используют DES.new() с ключом, сгенерированным через derive\_key, и режимом DES.MODE\_CFB. При шифровании генерируется случайная соль, которая добавляется к паролю перед хешированием, и сохраняется в начале зашифрованных данных. При расшифровке соль извлекается из начала файла и используется для восстановления ключа.

**Описание мест хранения зашифрованных данных**

Файл data/users.db.enc — основное хранилище зашифрованных данных. При запуске программы этот файл расшифровывается.

Файл data/users.db.tmp — временный файл, в который записывается расшифрованная база данных при работе программы. Удаляется при завершении работы.

Пароль для шифрования/дешифровки — вводится пользователем при запуске программы и при выходе из админки. С его помощью генерируется ключ шифрования.

При запуске: файл users.db.enc расшифровывается в users.db.tmp, из него загружается база данных. При расшифровке из начала файла извлекается соль, которая использовалась при шифровании, и с её помощью восстанавливается ключ.

При выходе: файл users.db.tmp шифруется с использованием новой соли, и записывается обратно в users.db.enc, после чего users.db.tmp удаляется.

Все операции с базой данных (изменение пользователей, паролей и т.д.) происходят с расшифрованной версией в users.db.tmp.

**Вывод**

Программа обеспечивает защиту данных за счёт шифрования файла базы данных и хранения паролей в виде хешей. Используются криптографические алгоритмы DES-CFB и MD4, как того требует задание. Всё работает через GUI на PyQt5, с поддержкой русского интерфейса и меню. Реализовано добавление случайного значения к ключу шифрования (соль).

**Листинг кода**

from PyQt5.QtWidgets import (

    QMainWindow, QVBoxLayout, QWidget, QPushButton, QListWidget, QMessageBox, QHBoxLayout, QInputDialog, QLineEdit, QMenuBar, QMenu, QAction, QStatusBar

)

class AdminWindow(QMainWindow):

    def \_\_init\_\_(self, um, parent\_window):

        # === Инициализация окна администратора ===

        super().\_\_init\_\_()

        # Храним UserManager

        self.um = um

        # Храним ссылку на родительское окно

        self.parent\_window = parent\_window

        # Устанавливаем заголовок окна

        self.setWindowTitle("Панель администратора")

        # Устанавливаем размеры окна

        self.setGeometry(300, 300, 900, 500)

        # Создаём меню

        self.create\_menu()

        # Создаём строку состояния

        self.statusBar().showMessage("Режим администратора")

        layout = QVBoxLayout()

        central\_widget = QWidget()

        central\_widget.setLayout(layout)

        self.setCentralWidget(central\_widget)

        self.user\_list = QListWidget()

        self.refresh\_user\_list()

        layout.addWidget(self.user\_list)

        btn\_layout = QHBoxLayout()

        btn\_add = QPushButton("Добавить пользователя")

        btn\_add.clicked.connect(self.add\_user)

        btn\_layout.addWidget(btn\_add)

        btn\_toggle\_lock = QPushButton("Переключить блокировку")

        btn\_toggle\_lock.clicked.connect(self.toggle\_lock)

        btn\_layout.addWidget(btn\_toggle\_lock)

        btn\_toggle\_policy = QPushButton("Переключить ограничения")

        btn\_toggle\_policy.clicked.connect(self.toggle\_policy)

        btn\_layout.addWidget(btn\_toggle\_policy)

        btn\_edit\_min\_length = QPushButton("Изменить мин. длину")

        btn\_edit\_min\_length.clicked.connect(self.edit\_min\_length)

        btn\_layout.addWidget(btn\_edit\_min\_length)

        btn\_edit\_valid\_months = QPushButton("Изменить срок действия (мес.)")

        btn\_edit\_valid\_months.clicked.connect(self.edit\_valid\_months)

        btn\_layout.addWidget(btn\_edit\_valid\_months)

        btn\_change\_admin\_password = QPushButton("Сменить пароль администратора")

        btn\_change\_admin\_password.clicked.connect(self.change\_admin\_password)

        btn\_layout.addWidget(btn\_change\_admin\_password)

        btn\_delete\_user = QPushButton("Удалить пользователя")

        btn\_delete\_user.clicked.connect(self.delete\_user)

        btn\_layout.addWidget(btn\_delete\_user)

        layout.addLayout(btn\_layout)

        # Кнопка "Выйти в главное меню"

        btn\_exit\_main = QPushButton("Выйти на страницу входа")

        btn\_exit\_main.clicked.connect(self.exit\_to\_main)

        layout.addWidget(btn\_exit\_main)

    def create\_menu(self):

        # === Создание меню окна ===

        menubar = self.menuBar()

        # Меню "Файл"

        file\_menu = menubar.addMenu('Файл')

        exit\_action = QAction('Выход из системы', self)

        exit\_action.triggered.connect(self.exit\_app)

        file\_menu.addAction(exit\_action)

        # Меню "Справка"

        help\_menu = menubar.addMenu('Справка')

        about\_action = QAction('О программе', self)

        about\_action.triggered.connect(self.show\_about)

        help\_menu.addAction(about\_action)

    def show\_about(self):

        # === Отображение информации о программе ===

        QMessageBox.about(self, "О программе",

            "Программа разграничения полномочий пользователей\n"

            "на основе парольной аутентификации c использованием встроенных криптопровайдеров.\n\n"

            "Автор: Малафеев Леонид Сергеевич. Студент гр. ПИбд-41.\n"

            "Задание: Вариант 19. Наличие букв и цифр.\n"

            "Шифрование: DES-CFB\n"

            "Хеширование: MD4")

    def refresh\_user\_list(self):

        # === Обновление списка пользователей в интерфейсе ===

        self.user\_list.clear()

        users = self.um.get\_all\_users()

        for user in users:

            username = user[0]

            pwd\_hash = user[1]

            locked = bool(user[2])

            policy = bool(user[3])

            min\_length = user[4]

            valid\_months = user[5]

            status = "Да" if locked else "Нет"

            policy\_status = "Да" if policy else "Нет"

            valid\_text = f"{valid\_months} месяцев" if valid\_months > 0 else "бессрочно"

            item\_text = (

                f"Пользователь: {username} | Заблокирован: {status} | "

                f"Ограничения: {policy\_status} | Мин. длина: {min\_length} | "

                f"Срок действия: {valid\_text}"

            )

            self.user\_list.addItem(item\_text)

    def add\_user(self):

        # === Добавление нового пользователя в базу данных ===

        name, ok = QInputDialog.getText(self, "Новый пользователь", "Имя:")

        # Проверка подтверждения ввода имени

        if not ok or not name:

            return

        cursor = self.um.conn.cursor()

        cursor.execute("SELECT username FROM Users WHERE username = ?", (name,))

        result = cursor.fetchone()

        # Проверка, существует ли пользователь с таким именем

        if result:

            QMessageBox.critical(self, "Ошибка", f"Пользователь с именем '{name}' уже существует.")

            return

        self.um.add\_user(name, '', False, False, 0, 0)

        self.refresh\_user\_list()

    def toggle\_policy(self):

        # === Переключение статуса ограничений для выбранного пользователя ===

        current = self.user\_list.currentItem()

        if current:

            full\_text = current.text()

            username\_part = full\_text.split(' | ')[0]

            username = username\_part.replace("Пользователь: ", "")

            self.um.toggle\_policy(username)

            self.refresh\_user\_list()

    def toggle\_lock(self):

        # === Переключение статуса блокировки для выбранного пользователя ===

        current = self.user\_list.currentItem()

        if current:

            full\_text = current.text()

            username\_part = full\_text.split(' | ')[0]

            username = username\_part.replace("Пользователь: ", "")

            self.um.toggle\_lock(username)

            self.refresh\_user\_list()

    def edit\_valid\_months(self):

        # === Изменение срока действия пароля для выбранного пользователя ===

        current = self.user\_list.currentItem()

        if current:

            full\_text = current.text()

            username\_part = full\_text.split(' | ')[0]

            username = username\_part.replace("Пользователь: ", "")

            cursor = self.um.conn.cursor()

            cursor.execute("SELECT valid\_months FROM Users WHERE username = ?", (username,))

            result = cursor.fetchone()

            if result:

                current\_months = result[0]

                months, ok = QInputDialog.getInt(self, "Срок действия пароля", "Введите количество месяцев (0 = бессрочно):", current\_months, 0, 120, 1)

                # Проверка, подтверждён ли ввод

                if ok:

                    cursor.execute("UPDATE Users SET valid\_months = ? WHERE username = ?", (months, username))

                    self.um.conn.commit()

                    self.refresh\_user\_list()

    def edit\_min\_length(self):

        # === Изменение минимальной длины пароля для выбранного пользователя ===

        current = self.user\_list.currentItem()

        if current:

            username = current.text().split(' | ')[0].replace("Пользователь: ", "")

            cursor = self.um.conn.cursor()

            cursor.execute("SELECT min\_length FROM Users WHERE username = ?", (username,))

            result = cursor.fetchone()

            if result:

                current\_length = result[0]

                length, ok = QInputDialog.getInt(self, "Мин. длина пароля", "Введите новую длину:", current\_length, 0, 100, 1)

                # Проверка, подтверждён ли ввод

                if ok:

                    cursor.execute("UPDATE Users SET min\_length = ? WHERE username = ?", (length, username))

                    self.um.conn.commit()

                    self.refresh\_user\_list()

    def delete\_user(self):

        # === Удаление выбранного пользователя из базы данных ===

        current = self.user\_list.currentItem()

        if current:

            username = current.text().split(' | ')[0]

            # Проверка, что не пытаемся удалить администратора

            if username == 'ADMIN':

                QMessageBox.critical(self, "Ошибка", "Нельзя удалить администратора!")

                return

            reply = QMessageBox.question(self, "Удалить пользователя", f"Удалить пользователя {username}?",

                                         QMessageBox.Yes | QMessageBox.No)

            # Русские кнопки

            yes\_button = reply.button(QMessageBox.Yes)

            if yes\_button:

                yes\_button.setText("Да")

            no\_button = reply.button(QMessageBox.No)

            if no\_button:

                no\_button.setText("Нет")

            # Проверка подтверждения удаления

            if reply == QMessageBox.Yes:

                try:

                    cursor = self.um.conn.cursor()

                    cursor.execute("DELETE FROM Users WHERE username = ?", (username,))

                    self.um.conn.commit()

                    self.refresh\_user\_list()

                    QMessageBox.information(self, "Успех", f"Пользователь {username} удалён.")

                except Exception as e:

                    QMessageBox.critical(self, "Ошибка", f"Ошибка при удалении: {str(e)}")

    def change\_admin\_password(self):

        # === Смена пароля администратора с проверкой старого пароля и ограничений ===

        from PyQt5.QtWidgets import QLineEdit

        old\_pwd, ok1 = QInputDialog.getText(self, "Старый пароль", "Введите старый пароль:", echo=QLineEdit.Password)

        # Проверка подтверждения ввода старого пароля

        if not ok1:

            return

        from Crypto.Hash import MD4

        h = MD4.new()

        h.update(old\_pwd.encode())

        old\_hash = h.hexdigest()

        cursor = self.um.conn.cursor()

        cursor.execute("SELECT password\_hash FROM Users WHERE username = ?", ("ADMIN",))

        result = cursor.fetchone()

        # Проверка, существует ли администратор

        if not result:

            QMessageBox.critical(self, "Ошибка", "Пользователь ADMIN не найден.")

            return

        pwd\_hash = result[0]

        # Проверка старого пароля: если в базе пустой — введённый должен быть пустым

        if pwd\_hash == '':

            if old\_pwd != '':

                QMessageBox.critical(self, "Ошибка", "Неверный старый пароль.")

                return

        else:

            # Если в базе не пустой — сравниваем хеши

            if pwd\_hash != old\_hash:

                QMessageBox.critical(self, "Ошибка", "Неверный старый пароль.")

                return

        new\_pwd, ok2 = QInputDialog.getText(self, "Новый пароль", "Введите новый пароль:", echo=QLineEdit.Password)

        # Проверка подтверждения ввода нового пароля

        if not ok2:

            return

        confirm\_pwd, ok3 = QInputDialog.getText(self, "Подтверждение", "Повторите новый пароль:", echo=QLineEdit.Password)

        # Проверка подтверждения ввода подтверждения пароля

        if not ok3:

            return

        # Проверка совпадения нового пароля и его подтверждения

        if new\_pwd != confirm\_pwd:

            QMessageBox.critical(self, "Ошибка", "Пароли не совпадают.")

            return

        # Проверка, включены ли ограничения для администратора

        cursor.execute("SELECT password\_policy FROM Users WHERE username = ?", ("ADMIN",))

        result = cursor.fetchone()

        if result and result[0]:

            # Если ограничения включены — проверяем, содержит ли пароль буквы и цифры

            if not self.um.check\_password\_policy(new\_pwd):

                reply = QMessageBox.question(self, "Ошибка",

                                             f"Пароль не соответствует требованиям: должен содержать буквы и цифры.\n"

                                             "Повторить ввод?",

                                             QMessageBox.Yes | QMessageBox.No,

                                             QMessageBox.Yes)

                # Русские кнопки

                yes\_button = reply.button(QMessageBox.Yes)

                if yes\_button:

                    yes\_button.setText("Да")

                no\_button = reply.button(QMessageBox.No)

                if no\_button:

                    no\_button.setText("Нет")

                # Проверка подтверждения повторного ввода

                if reply == QMessageBox.Yes:

                    self.change\_admin\_password()

                return

        # Проверка минимальной длины пароля

        cursor.execute("SELECT min\_length FROM Users WHERE username = ?", ("ADMIN",))

        result = cursor.fetchone()

        if result and result[0] > 0:

            # Извлекаем минимальную длину

            min\_len = result[0]

            # Проверка, достаточно ли длинный пароль

            if len(new\_pwd) < min\_len:

                reply = QMessageBox.question(self, "Ошибка",

                                             f"Пароль не соответствует требованиям: минимальная длина — {min\_len} символов.\n"

                                             "Повторить ввод?",

                                             QMessageBox.Yes | QMessageBox.No,

                                             QMessageBox.Yes)

                # Русские кнопки

                yes\_button = reply.button(QMessageBox.Yes)

                if yes\_button:

                    yes\_button.setText("Да")

                no\_button = reply.button(QMessageBox.No)

                if no\_button:

                    no\_button.setText("Нет")

                # Проверка подтверждения повторного ввода

                if reply == QMessageBox.Yes:

                    self.change\_admin\_password()

                return

        # === Хеширование нового пароля и обновление в базе данных ===

        h = MD4.new()

        h.update(new\_pwd.encode())

        new\_hash = h.hexdigest()

        cursor.execute("UPDATE Users SET password\_hash = ? WHERE username = ?", (new\_hash, "ADMIN"))

        self.um.conn.commit()

        QMessageBox.information(self, "Успех", "Пароль администратора изменён.")

    def exit\_to\_main(self):

        # === Шифрование базы данных и возврат на главное окно ===

        pwd, ok = QInputDialog.getText(self, "Шифрование", "Введите пароль для шифрования файла:", echo=QLineEdit.Password)

        # Проверка подтверждения ввода пароля

        if not ok or not pwd:

            return

        from crypto import encrypt\_file

        with open(self.um.db\_path, 'rb') as f:

            data = f.read()

        encrypted = encrypt\_file(data, pwd)

        with open(self.um.enc\_file, 'wb') as f:

            f.write(encrypted)

        self.close()

        self.parent\_window.show()

    def exit\_app(self):

        # === Завершение работы приложения с шифрованием базы данных ===

        pwd, ok = QInputDialog.getText(self, "Шифрование", "Введите пароль для шифрования файла:", echo=QLineEdit.Password)

        # Проверка подтверждения ввода пароля

        if not ok or not pwd:

            return

        from crypto import encrypt\_file

        with open(self.um.db\_path, 'rb') as f:

            data = f.read()

        encrypted = encrypt\_file(data, pwd)

        with open(self.um.enc\_file, 'wb') as f:

            f.write(encrypted)

        self.um.close()

        self.close()

import hashlib

from Crypto.Cipher import DES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

from Crypto.Hash import MD4

import os

def derive\_key(password: str, salt: bytes) -> bytes:

    # === Генерация ключа из пароля с использованием соли ===

    h = MD4.new()

    h.update(salt + password.encode())

    return h.digest()[:8]

def encrypt\_file(data: bytes, password: str) -> bytes:

    # === Шифрование данных с использованием DES-CFB и случайной соли ===

    salt = get\_random\_bytes(8)

    key = derive\_key(password, salt)

    iv = get\_random\_bytes(8)

    cipher = DES.new(key, DES.MODE\_CFB, iv=iv)

    encrypted = cipher.encrypt(pad(data, DES.block\_size))

    return salt + iv + encrypted

def decrypt\_file(data: bytes, password: str) -> bytes:

    # === Расшифровка данных с использованием DES-CFB и соли из начала файла ===

    salt = data[:8]

    iv = data[8:16]

    encrypted = data[16:]

    key = derive\_key(password, salt)

    cipher = DES.new(key, DES.MODE\_CFB, iv=iv)

    decrypted = unpad(cipher.decrypt(encrypted), DES.block\_size)

    return decrypted

from PyQt5.QtWidgets import (

    QApplication, QMainWindow, QVBoxLayout, QWidget, QLabel, QLineEdit, QPushButton, QMessageBox

)

from user\_manager import UserManager

class LoginWindow(QMainWindow):

    def \_\_init\_\_(self, db\_password, main\_window):

        # === Инициализация окна входа ===

        super().\_\_init\_\_()

        self.main\_window = main\_window

        self.db\_password = db\_password

        self.um = UserManager(password=self.db\_password)

        self.setWindowTitle("Вход")

        self.setGeometry(300, 300, 300, 150)

        # Счётчик попыток

        self.attempts = 0

        # Максимальное количество попыток

        self.max\_attempts = 3

        layout = QVBoxLayout()

        central\_widget = QWidget()

        central\_widget.setLayout(layout)

        self.setCentralWidget(central\_widget)

        self.username\_input = QLineEdit()

        self.password\_input = QLineEdit()

        self.password\_input.setEchoMode(QLineEdit.Password)

        layout.addWidget(QLabel("Имя:"))

        layout.addWidget(self.username\_input)

        layout.addWidget(QLabel("Пароль:"))

        layout.addWidget(self.password\_input)

        btn = QPushButton("Войти")

        btn.clicked.connect(self.login)

        layout.addWidget(btn)

        back\_btn = QPushButton("Выйти на главную")

        back\_btn.clicked.connect(self.go\_back\_to\_main)

        layout.addWidget(back\_btn)

    def login(self):

        # === Обработка попытки входа пользователя ===

        username = self.username\_input.text()

        password = self.password\_input.text()

        success, msg = self.um.authenticate(username, password)

        if success:

            # Сброс попыток при успешном входе

            self.attempts = 0

            if username == "ADMIN":

                from admin\_window import AdminWindow

                self.admin\_window = AdminWindow(self.um, self)

                self.admin\_window.show()

                self.hide()

            else:

                from user\_window import UserWindow

                self.user\_window = UserWindow(self.um, username, self)

                self.user\_window.show()

                self.hide()

        else:

            # Если пользователь не найден — сбрасываем счётчик

            if "Пользователь не найден" in msg:

                reply = self.show\_custom\_question("Ошибка", f"{msg}. Повторить ввод?")

                # Проверка подтверждения повторного ввода

                if reply == QMessageBox.Yes:

                    self.username\_input.clear()

                    self.password\_input.clear()

                else:

                    self.close()

            else:

                # Увеличиваем счётчик при неправильном пароле

                self.attempts += 1

                remaining = self.max\_attempts - self.attempts

                # Проверка, есть ли ещё попытки

                if remaining > 0:

                    QMessageBox.critical(self, "Ошибка", f"Неверный пароль, осталось попыток: {remaining}")

                    self.password\_input.clear()

                else:

                    QMessageBox.critical(self, "Ошибка", "Превышено количество попыток. Программа завершена.")

                    self.close()

                    import sys

                    sys.exit()

    def go\_back\_to\_main(self):

        # === Возврат на главное окно ===

        self.close()

        self.main\_window.show()

    def show\_custom\_question(self, title, text):

        # === Отображение диалога с русскими кнопками ===

        msg\_box = QMessageBox(self)

        msg\_box.setWindowTitle(title)

        msg\_box.setText(text)

        msg\_box.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

        msg\_box.setDefaultButton(QMessageBox.Yes)

        # Русские надписи на кнопках

        yes\_button = msg\_box.button(QMessageBox.Yes)

        if yes\_button:

            yes\_button.setText("Да")

        no\_button = msg\_box.button(QMessageBox.No)

        if no\_button:

            no\_button.setText("Нет")

        return msg\_box.exec\_()

import sys

import os

from PyQt5.QtWidgets import QApplication, QMainWindow, QMenuBar, QMenu, QAction, QStatusBar, QPushButton, QVBoxLayout, QWidget, QMessageBox, QLineEdit, QInputDialog

from PyQt5.QtCore import Qt

from login\_window import LoginWindow

from password\_dialog import PasswordDialog

class MainWindow(QMainWindow):

    def \_\_init\_\_(self, db\_password):

        # === Инициализация главного окна приложения ===

        super().\_\_init\_\_()

        # Храним пароль для расшифровки базы данных

        self.db\_password = db\_password

        # Устанавливаем заголовок окна

        self.setWindowTitle("Менеджер пользователей")

        # Устанавливаем размеры окна

        self.setGeometry(100, 100, 800, 600)

        # Создаём меню

        self.create\_menu()

        # Создаём строку состояния

        self.statusBar().showMessage("Готов")

        # Центральный виджет с кнопкой "Войти"

        central\_widget = QWidget()

        layout = QVBoxLayout()

        central\_widget.setLayout(layout)

        # Создаём кнопку, не растягивая на всю ширину

        self.login\_button = QPushButton("Войти")

        self.login\_button.setMaximumWidth(200)  # Устанавливаем максимальную ширину

        self.login\_button.clicked.connect(self.open\_login)

        layout.addWidget(self.login\_button)

        layout.setAlignment(Qt.AlignCenter)  # Центрируем содержимое layout

        self.setCentralWidget(central\_widget)

    def create\_menu(self):

        # === Создание меню окна ===

        menubar = self.menuBar()

        # Меню "Файл"

        file\_menu = menubar.addMenu('Файл')

        exit\_action = QAction('Выход из системы', self)

        exit\_action.triggered.connect(self.exit\_app)

        file\_menu.addAction(exit\_action)

        # Меню "Справка"

        help\_menu = menubar.addMenu('Справка')

        about\_action = QAction('О программе', self)

        about\_action.triggered.connect(self.show\_about)

        help\_menu.addAction(about\_action)

    def show\_about(self):

        # === Отображение информации о программе ===

        QMessageBox.about(self, "О программе",

            "Программа разграничения полномочий пользователей\n"

            "на основе парольной аутентификации c использованием встроенных криптопровайдеров.\n\n"

            "Автор: Малафеев Леонид Сергеевич. Студент гр. ПИбд-41.\n"

            "Задание: Вариант 19. Наличие букв и цифр.\n"

            "Шифрование: DES-CFB\n"

            "Хеширование: MD4")

    def open\_login(self):

        # === Открытие окна входа ===

        self.login\_window = LoginWindow(self.db\_password, self)

        self.login\_window.show()

        self.hide()

    def exit\_app(self):

        # === Завершение работы приложения с шифрованием базы данных ===

        if hasattr(self, 'login\_window') and self.login\_window and self.login\_window.um:

            pwd, ok = QInputDialog.getText(self, "Шифрование", "Введите пароль для шифрования файла:", echo=QLineEdit.Password)

            if not ok or not pwd:

                return

            from crypto import encrypt\_file

            with open(self.login\_window.um.db\_path, 'rb') as f:

                data = f.read()

            encrypted = encrypt\_file(data, pwd)

            with open(self.login\_window.um.enc\_file, 'wb') as f:

                f.write(encrypted)

            self.login\_window.um.close()

        self.close()

if \_\_name\_\_ == "\_\_main\_\_":

    # === Запуск приложения ===

    app = QApplication(sys.argv)

    # --- Авторизация БД при запуске ---

    if os.path.exists('data/users.db.enc'):

        # Проверка существования зашифрованного файла базы данных

        pwd\_dialog = PasswordDialog("Расшифровка", "Введите пароль для расшифровки файла:")

    else:

        # Если файл не существует — создаём новый

        pwd\_dialog = PasswordDialog("Шифрование", "Введите пароль для шифрования файла:")

    pwd = pwd\_dialog.get\_password()

    # Проверка подтверждения ввода пароля

    if not pwd:

        sys.exit()

    main\_window = MainWindow(pwd)

    main\_window.show()

    def on\_exit():

        # === Закрытие соединения с базой данных при выходе ===

        if hasattr(main\_window, 'login\_window') and main\_window.login\_window:

            main\_window.login\_window.um.close()

    app.aboutToQuit.connect(on\_exit)

    sys.exit(app.exec\_())

from PyQt5.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit, QPushButton, QApplication

from PyQt5.QtCore import Qt

class PasswordDialog(QDialog):

    def \_\_init\_\_(self, title="Введите пароль", message="Пароль:"):

        # === Инициализация диалога ввода пароля ===

        super().\_\_init\_\_()

        self.setWindowTitle(title)

        self.setModal(True)

        self.resize(300, 100)

        layout = QVBoxLayout()

        self.setLayout(layout)

        self.label = QLabel(message)

        layout.addWidget(self.label)

        self.password\_input = QLineEdit()

        self.password\_input.setEchoMode(QLineEdit.Password)

        layout.addWidget(self.password\_input)

        btn\_layout = QVBoxLayout()

        self.ok\_button = QPushButton("ОК")

        self.ok\_button.clicked.connect(self.accept)

        btn\_layout.addWidget(self.ok\_button)

        self.cancel\_button = QPushButton("Отмена")

        self.cancel\_button.clicked.connect(self.reject)

        btn\_layout.addWidget(self.cancel\_button)

        layout.addLayout(btn\_layout)

        self.password = None

    def get\_password(self):

        # === Получение введённого пароля из диалога ===

        result = self.exec\_()

        # Проверка подтверждения ввода

        if result == QDialog.Accepted:

            self.password = self.password\_input.text()

        return self.password

import sqlite3

import os

from datetime import datetime, timedelta

from crypto import decrypt\_file, encrypt\_file

from Crypto.Hash import MD4

class UserManager:

    def \_\_init\_\_(self, enc\_file='data/users.db.enc', password=None):

        # === Инициализация менеджера пользователей ===

        os.makedirs(os.path.dirname(enc\_file), exist\_ok=True)

        self.enc\_file = enc\_file

        self.db\_path = 'data/users.db.tmp'

        self.conn = None

        self.load\_from\_encrypted(password)

    def load\_from\_encrypted(self, pwd):

        # === Загрузка и расшифровка базы данных из зашифрованного файла ===

        if not os.path.exists(self.enc\_file):

            # Если файл не существует — создаём новый с администратором

            self.init\_db()

            self.add\_user('ADMIN', '', False, False, 0, 0)

            self.save\_to\_encrypted(pwd)

            return

        try:

            with open(self.enc\_file, 'rb') as f:

                data = f.read()

            decrypted = decrypt\_file(data, pwd)

            with open(self.db\_path, 'wb') as f:

                f.write(decrypted)

            self.conn = sqlite3.connect(self.db\_path)

            # Проверка наличия администратора в расшифрованной базе

            cursor = self.conn.cursor()

            cursor.execute("SELECT username FROM Users WHERE username = 'ADMIN'")

            result = cursor.fetchone()

            # Проверка, найден ли администратор

            if not result:

                from PyQt5.QtWidgets import QMessageBox

                QMessageBox.critical(None, "Ошибка", "Ошибка расшифровки. Файл повреждён или зашифрован другим паролем.")

                raise SystemExit

        except Exception as e:

            from PyQt5.QtWidgets import QMessageBox

            QMessageBox.critical(None, "Ошибка", "Ошибка расшифровки. Неверный пароль или файл поврежден.")

            raise SystemExit

    def init\_db(self):

        # === Инициализация структуры базы данных ===

        self.conn = sqlite3.connect(self.db\_path)

        cursor = self.conn.cursor()

        cursor.execute('''

            CREATE TABLE IF NOT EXISTS Users (

                username TEXT PRIMARY KEY,

                password\_hash TEXT,

                locked INTEGER,

                password\_policy INTEGER,

                min\_length INTEGER,

                valid\_months INTEGER

            )

        ''')

        self.conn.commit()

    def save\_to\_encrypted(self, pwd):

        # === Шифрование и сохранение базы данных в файл ===

        with open(self.db\_path, 'rb') as f:

            data = f.read()

        encrypted = encrypt\_file(data, pwd)

        with open(self.enc\_file, 'wb') as f:

            f.write(encrypted)

    def authenticate(self, username, password):

        # === Аутентификация пользователя по имени и паролю ===

        cursor = self.conn.cursor()

        cursor.execute("SELECT password\_hash, locked, valid\_months FROM Users WHERE username = ?", (username,))

        row = cursor.fetchone()

        # Проверка, существует ли пользователь

        if not row:

            return False, "Пользователь не найден"

        pwd\_hash, locked, valid\_months = row

        # Проверка, заблокирован ли пользователь

        if locked:

            return False, "Аккаунт заблокирован"

        # Если в базе пароль пустой — проверяем, что введённый тоже пустой

        if pwd\_hash == '':

            if password == '':

                return True, "Успешно"

            else:

                return False, "Неверный пароль"

        else:

            # Если в базе пароль не пустой — хешируем введённый и сравниваем

            h = MD4.new()

            h.update(password.encode())

            input\_hash = h.hexdigest()

            # Проверка совпадения хешей

            if pwd\_hash != input\_hash:

                return False, "Неверный пароль"

        # Проверка срока действия (реализация упрощена)

        if valid\_months > 0:

            pass

        return True, "Успешно"

    def check\_password\_policy(self, password):

        # === Проверка, содержит ли пароль буквы и цифры ===

        has\_letter = any(c.isalpha() for c in password)

        has\_digit = any(c.isdigit() for c in password)

        return has\_letter and has\_digit

    def add\_user(self, username, password\_hash, locked, password\_policy, min\_length, valid\_months):

        # === Добавление или обновление пользователя в базе данных ===

        cursor = self.conn.cursor()

        cursor.execute('''

            INSERT OR REPLACE INTO Users VALUES (?, ?, ?, ?, ?, ?)

        ''', (username, password\_hash, int(locked), int(password\_policy), min\_length, valid\_months))

        self.conn.commit()

    def get\_all\_users(self):

        # === Получение всех пользователей из базы данных ===

        cursor = self.conn.cursor()

        cursor.execute("SELECT \* FROM Users")

        return cursor.fetchall()

    def toggle\_lock(self, username):

        # === Переключение статуса блокировки пользователя ===

        cursor = self.conn.cursor()

        cursor.execute("SELECT locked FROM Users WHERE username = ?", (username,))

        row = cursor.fetchone()

        if row:

            new\_status = 1 - row[0]

            cursor.execute("UPDATE Users SET locked = ? WHERE username = ?", (new\_status, username))

            self.conn.commit()

    def toggle\_policy(self, username):

        # === Переключение статуса ограничений пароля пользователя ===

        cursor = self.conn.cursor()

        cursor.execute("SELECT password\_policy FROM Users WHERE username = ?", (username,))

        row = cursor.fetchone()

        if row:

            new\_status = 1 - row[0]

            cursor.execute("UPDATE Users SET password\_policy = ? WHERE username = ?", (new\_status, username))

            self.conn.commit()

    def change\_password(self, username, old\_password, new\_password, confirm\_password):

        # === Изменение пароля пользователя с проверками ===

        cursor = self.conn.cursor()

        cursor.execute("SELECT password\_hash, password\_policy, min\_length FROM Users WHERE username = ?", (username,))

        row = cursor.fetchone()

        # Проверка, существует ли пользователь

        if not row:

            return False, "Пользователь не найден"

        pwd\_hash, policy\_enabled, min\_length = row

        # Проверка старого пароля

        if pwd\_hash == '':

            # Если в базе пароль пустой — введённый должен быть пустым

            if old\_password != '':

                return False, "Старый пароль неверен"

        else:

            # Если в базе пароль не пустой — хешируем введённый и сравниваем

            h = MD4.new()

            h.update(old\_password.encode())

            old\_hash = h.hexdigest()

            # Проверка совпадения хешей

            if pwd\_hash != old\_hash:

                return False, "Старый пароль неверен"

        # Проверка совпадения нового пароля и его подтверждения

        if new\_password != confirm\_password:

            return False, "Пароли не совпадают"

        # Проверка, включены ли ограничения (буквы и цифры)

        if policy\_enabled:

            # Проверка, содержит ли пароль буквы и цифры

            if not self.check\_password\_policy(new\_password):

                return False, f"Пароль не соответствует требованиям: должен содержать буквы и цифры."

        # Проверка минимальной длины пароля

        if len(new\_password) < min\_length:

            return False, f"Пароль не соответствует требованиям: минимальная длина — {min\_length} символов."

        # === Хеширование нового пароля и обновление в базе ===

        h = MD4.new()

        h.update(new\_password.encode())

        new\_hash = h.hexdigest()

        cursor.execute("UPDATE Users SET password\_hash = ? WHERE username = ?", (new\_hash, username))

        self.conn.commit()

        return True, "Пароль изменен"

    def close(self):

        # === Закрытие соединения с базой данных и удаление временного файла ===

        if self.conn:

            self.conn.close()

        if os.path.exists(self.db\_path):

            os.remove(self.db\_path)

from PyQt5.QtWidgets import (

    QMainWindow, QVBoxLayout, QWidget, QPushButton, QMessageBox, QLineEdit, QInputDialog, QHBoxLayout, QMenuBar, QMenu, QAction, QStatusBar

)

class UserWindow(QMainWindow):

    def \_\_init\_\_(self, um, username, parent\_window=None):

        # === Инициализация окна пользователя ===

        super().\_\_init\_\_()

        # Храним UserManager

        self.um = um

        # Храним имя пользователя

        self.username = username

        # Храним ссылку на родительское окно

        self.parent\_window = parent\_window

        # Устанавливаем заголовок окна

        self.setWindowTitle(f"Пользователь: {username}")

        # Устанавливаем размеры окна

        self.setGeometry(300, 300, 300, 200)

        # Создаём меню

        self.create\_menu()

        # Создаём строку состояния

        self.statusBar().showMessage(f"Пользователь: {username}")

        layout = QVBoxLayout()

        central\_widget = QWidget()

        central\_widget.setLayout(layout)

        self.setCentralWidget(central\_widget)

        btn\_change = QPushButton("Сменить пароль")

        btn\_change.clicked.connect(self.change\_password)

        layout.addWidget(btn\_change)

        btn\_exit\_main = QPushButton("Выйти на страницу входа")

        btn\_exit\_main.clicked.connect(self.exit\_to\_main)

        layout.addWidget(btn\_exit\_main)

    def create\_menu(self):

        # === Создание меню окна ===

        menubar = self.menuBar()

        # Меню "Файл"

        file\_menu = menubar.addMenu('Файл')

        exit\_action = QAction('Выход из системы', self)

        exit\_action.triggered.connect(self.exit\_app)

        file\_menu.addAction(exit\_action)

        # Меню "Справка"

        help\_menu = menubar.addMenu('Справка')

        about\_action = QAction('О программе', self)

        about\_action.triggered.connect(self.show\_about)

        help\_menu.addAction(about\_action)

    def show\_about(self):

        # === Отображение информации о программе ===

        from PyQt5.QtWidgets import QMessageBox

        QMessageBox.about(self, "О программе",

            "Программа разграничения полномочий пользователей\n"

            "на основе парольной аутентификации c использованием встроенных криптопровайдеров.\n\n"

            "Автор: Малафеев Леонид Сергеевич. Студент гр. ПИбд-41.\n"

            "Задание: Вариант 19. Наличие букв и цифр.\n"

            "Шифрование: DES-CFB\n"

            "Хеширование: MD4")

    def change\_password(self):

        # === Смена пароля пользователя с проверками ===

        from PyQt5.QtWidgets import QLineEdit

        old\_pwd, ok1 = self.get\_text\_input("Старый пароль", "Введите старый пароль:", QLineEdit.Password)

        # Проверка подтверждения ввода старого пароля

        if not ok1:

            return

        new\_pwd, ok2 = self.get\_text\_input("Новый пароль", "Введите новый пароль:", QLineEdit.Password)

        # Проверка подтверждения ввода нового пароля

        if not ok2:

            return

        confirm\_pwd, ok3 = self.get\_text\_input("Подтверждение", "Повторите новый пароль:", QLineEdit.Password)

        # Проверка подтверждения ввода подтверждения

        if not ok3:

            return

        success, msg = self.um.change\_password(self.username, old\_pwd, new\_pwd, confirm\_pwd)

        # Проверка, успешна ли смена пароля

        if not success:

            # Если ошибка — предоставляем выбор

            reply = QMessageBox.question(self, "Ошибка",

                                         f"{msg}\n"

                                         "Повторить ввод?",

                                         QMessageBox.Yes | QMessageBox.No)

            # Русские кнопки

            yes\_button = reply.button(QMessageBox.Yes)

            if yes\_button:

                yes\_button.setText("Да")

            no\_button = reply.button(QMessageBox.No)

            if no\_button:

                no\_button.setText("Нет")

            # Проверка подтверждения повторного ввода

            if reply == QMessageBox.Yes:

                # Повторить смену пароля

                self.change\_password()

            return

        else:

            QMessageBox.information(self, "Результат", msg)

    def exit\_to\_main(self):

        # === Шифрование базы данных и возврат на главное окно ===

        pwd, ok = self.get\_text\_input("Шифрование", "Введите пароль для шифрования файла:", QLineEdit.Password)

        # Проверка подтверждения ввода пароля

        if not ok or not pwd:

            return

        from crypto import encrypt\_file

        with open(self.um.db\_path, 'rb') as f:

            data = f.read()

        encrypted = encrypt\_file(data, pwd)

        with open(self.um.enc\_file, 'wb') as f:

            f.write(encrypted)

        self.close()

        if self.parent\_window:

            self.parent\_window.show()

    def exit\_app(self):

        # === Завершение работы приложения с шифрованием базы данных ===

        pwd, ok = self.get\_text\_input("Шифрование", "Введите пароль для шифрования файла:", QLineEdit.Password)

        # Проверка подтверждения ввода пароля

        if not ok or not pwd:

            return

        from crypto import encrypt\_file

        with open(self.um.db\_path, 'rb') as f:

            data = f.read()

        encrypted = encrypt\_file(data, pwd)

        with open(self.um.enc\_file, 'wb') as f:

            f.write(encrypted)

        self.um.close()

        self.close()

    def get\_text\_input(self, title, label, echo\_mode):

        # === Получение текстового ввода с русскими кнопками ===

        dialog = QInputDialog(self)

        dialog.setWindowTitle(title)

        dialog.setLabelText(label)

        dialog.setTextEchoMode(echo\_mode)

        # Русские кнопки

        ok\_button = dialog.findChild(QPushButton)

        if ok\_button:

            ok\_button.setText("ОК")

        cancel\_button = dialog.findChild(QPushButton, "cancel")

        if cancel\_button:

            cancel\_button.setText("Отмена")

        if dialog.exec\_() == QInputDialog.Accepted:

            return dialog.textValue(), True

        else:

            return "", False