МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

Связывание приложения на Python с базой данных под управлением PostgreSQL.

Отчёт по лабораторной работе №5
по дисциплине
«Управление данными»

Разработал студент гр. ИВТб-2301-05-00 _____/Черкасов А. А./
(подпись)

Старший Преподователь _____/Клюкин В. Л./
(подпись)

Работа защищена «__»_____ 2025 г.

Киров
2025

Цели лабораторной работы

- познакомиться с библиотекой в языке Python для связывания приложения с БД;
- освоить на практике основы взаимодействия с БД под управлением PostgreSQL в приложении на Python.

Задание

Создать приложение с графическим интерфейсом на языке Python, использующее БД, разработанную в предыдущих лабораторных работах, со следующими требованиями:

1. Названия колонок, кнопок, объектов ввода/вывода на русском языке.
2. Запрет ввода отрицательных значений.
3. Ввод данных для выборки регистронезависимый (используются функции UPPER или LOWER).
4. Для любой таблицы с внешним ключом реализовать:
    - вывод, удаление и изменение данных таблицы;
    - проверку ввода уже имеющихся данных с выводом сообщения пользователю;
    - удаление при подтверждении;
    - выполнение фильтра (выборки) по значениям строк.
5. При добавлении новой строки внешний ключ выбирается из списка значений родительской таблицы.
6. Сохранение или удаление строки реализовано с помощью функции PL/pgSQL.
7. Фильтрация значений при поиске производится через запрос, а не в полученной коллекции.

Реализация приложения

Диаграмма классов

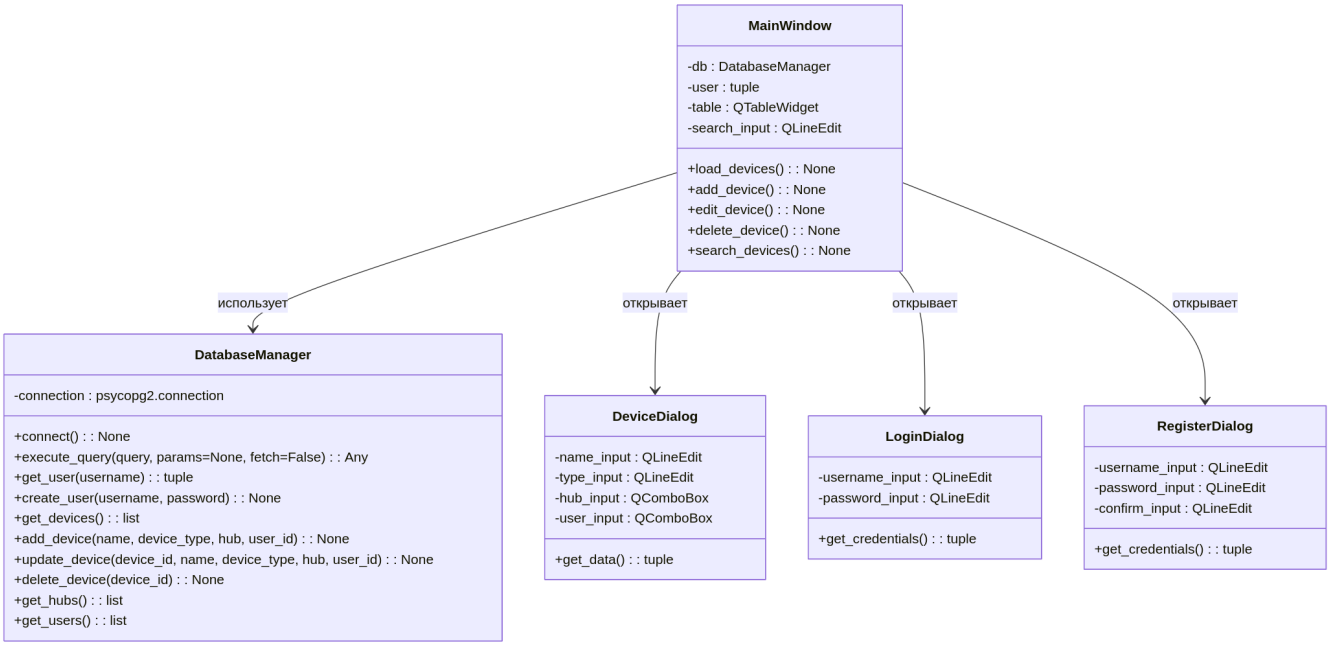

Рис. 1: Диаграмма классов приложения

Диаграмма классов показывает архитектуру приложения и организацию работы с базой данных:

- `Database` — класс для подключения и выполнения операций с PostgreSQL
- `MainWindow` — главное окно приложения с пользовательским интерфейсом
- `LoginDialog` — диалог аутентификации пользователей
- `RegisterDialog` — диалог регистрации новых пользователей
- `DeviceDialog` — диалог управления устройствами
- `SearchLineEdit` — кастомное поле поиска с поддержкой Esc

Подключение к базе данных

Класс Database

Центральный класс для работы с PostgreSQL, реализующий паттерн Singleton для управления соединением:

```
import psycopg2
from psycopg2 import sql
import hashlib
import os

class Database:
  def __init__(self):
    # Установка соединения с PostgreSQL
    self.conn = psycopg2.connect(
      host="localhost",
      port=5433,
      database="pozordom",
      user="pozordom_user",
      password="pozordom_pass"
    )
    self.conn.autocommit = True
```

Обработка исключений

Реализована обработка всех типов ошибок подключения:

```
def connect_to_database(self):
  try:
    self.db = Database()
    self.statusBar().showMessage("Подключено к базе данных")
  except Exception as e:
    QMessageBox.critical(self, "Ошибка подключения",
      f"Не удалось подключиться к базе данных: {str(e)}")
```

Выполнение запросов к БД

Чтение данных

Получение данных из таблиц с использованием параметризованных запросов:

```
def get_devices_for_user(self, user_id, filter_text=""):
  query = """
    SELECT d.id, d.name, dt.type_name, h.name AS hub_name
    FROM devices d
    JOIN device_types dt ON d.type_id = dt.id
    JOIN hubs h ON d.hub_id = h.id
    WHERE h.user_id = %s AND d.name ILIKE %s
    ORDER BY d.id
  """
  with self.conn.cursor() as cur:
    cur.execute(query, (user_id, f"%{filter_text}%"))
      return cur.fetchall()
```

Вставка и обновление данных

Использование функции PL/pgSQL для безопасного сохранения:

```
def save_device(self, device_id, hub_id, type_id, name, status):
  with self.conn.cursor() as cur:
    cur.callproc('save_devices', [device_id, hub_id, type_id, name, status])
    return cur.fetchone()[0]
```

Удаление данных

Безопасное удаление с очисткой связанных записей:

```
def delete_device_safe(self, device_id):
  with self.conn.cursor() as cur:
    cur.execute("DELETE FROM log_devices WHERE device_id = %s", (device_id,))
    cur.execute("DELETE FROM devices WHERE id = %s", (device_id,))
```

Работа с внешними ключами

Загрузка связанных данных

При добавлении устройств внешние ключи выбираются из списков:

```
def load_data(self):
    hubs = self.db.get_user_hubs(self.user_id)
    self.hub_combo.clear()
    for hub_id, hub_name in hubs:
        self.hub_combo.addItem(hub_name, hub_id)

    device_types = self.db.get_device_types()
    self.type_combo.clear()
    for type_id, type_name in device_types:
        self.type_combo.addItem(type_name, type_id)
```

Валидация уникальности

Проверка отсутствия дубликатов перед сохранением:

```
def device_exists(self, name, exclude_id=None):
  query = "SELECT 1 FROM devices WHERE name = %s"
  params = [name]
  if exclude_id:
    query += " AND id != %s"
    params.append(exclude_id)
  with self.conn.cursor() as cur:
    cur.execute(query, params)
    return cur.fetchone() is not None
```

Ручной поиск с фильтрацией

Регистронезависимый поиск

Использование ILIKE для регистронезависимого поиска:

```
def perform_search(self):
```

```
  filter_text = self.filter_edit.text().strip()
  devices = self.db.get_devices_for_user(self.current_user_id, filter_text)

  if not devices and filter_text:
    QMessageBox.warning(self, "Поиск",
      f"Устройства с названием '{filter_text}' не найдены")
    return
```

Подтверждение удаления

Диалог подтверждения перед удалением устройств:

```
reply = QMessageBox.question(
  self, "Подтверждение удаления",
  f"Вы действительно хотите удалить устройство '{device_name}'?",
  QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
  QMessageBox.StandardButton.No
)
```

Скриншоты интерфейса

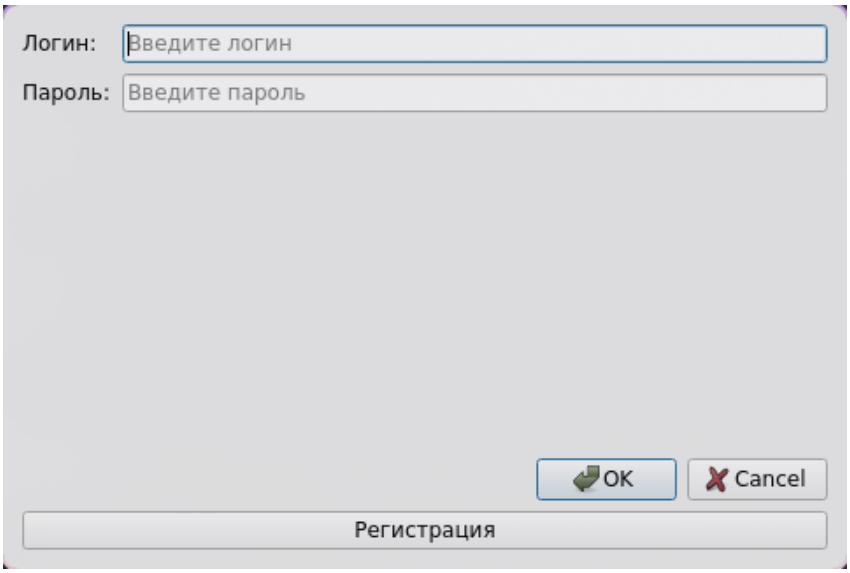

Рисунок 1 - Окно входа в систему

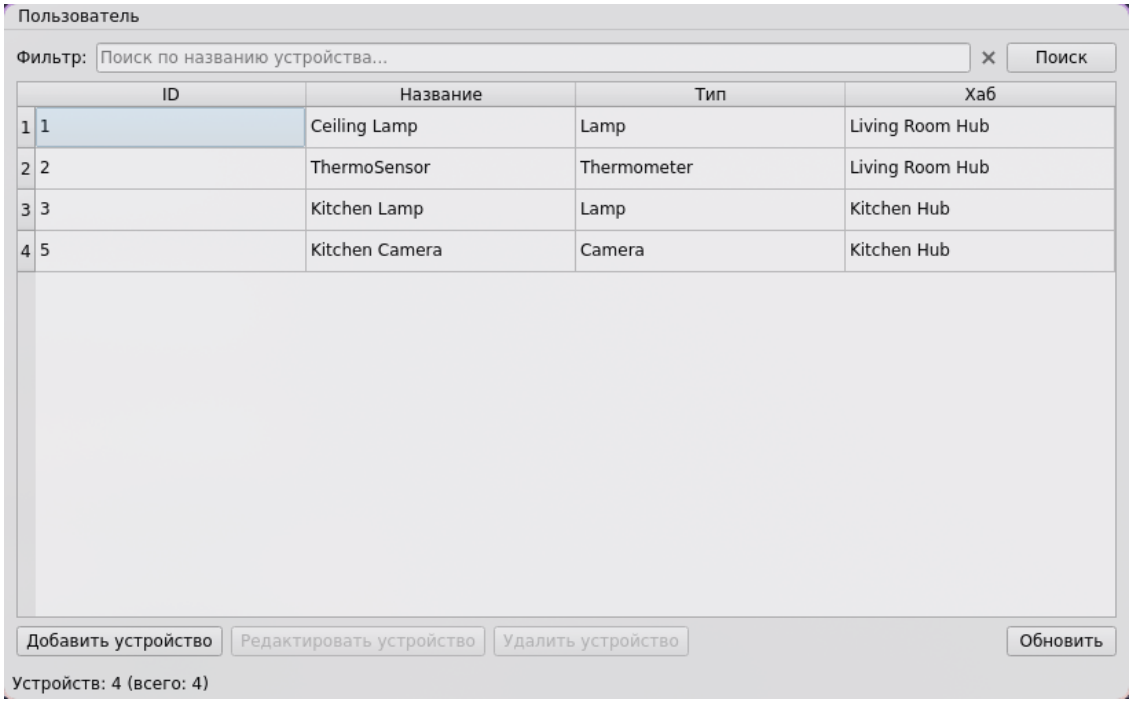

Рисунок 2 - Главное окно приложения

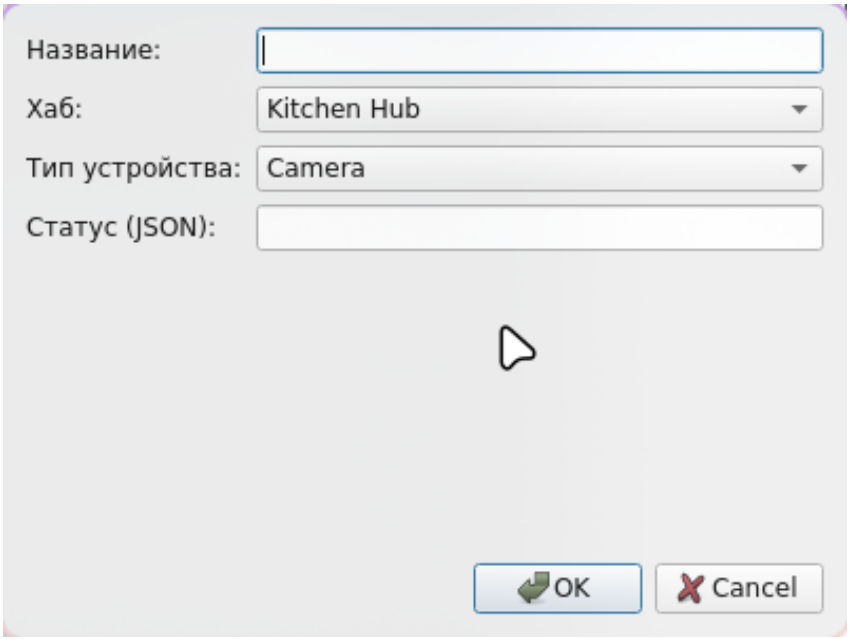

Рисунок 3 - Диалог добавления устройства

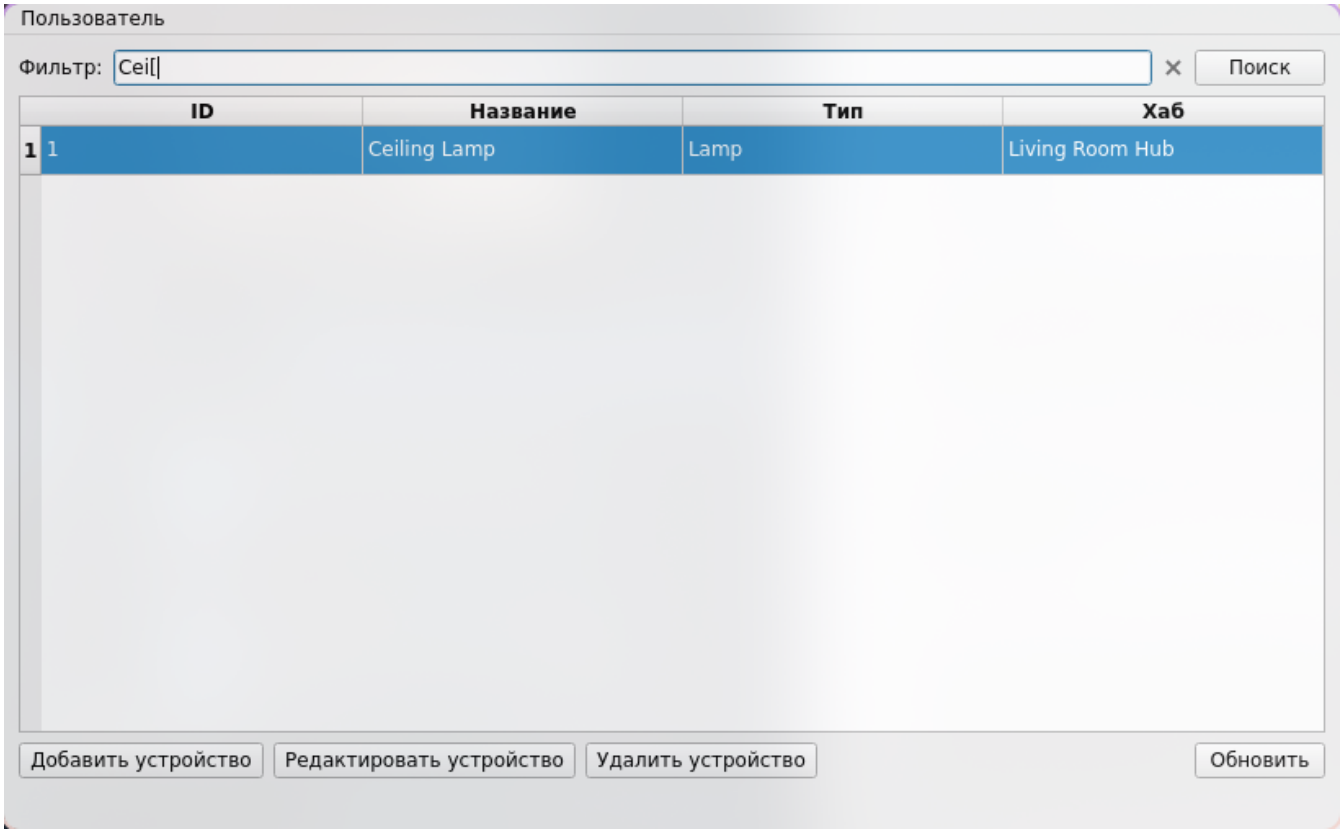

Рисунок 4 - Функционал ручного поиска

Вывод

В ходе выполнения лабораторной работы №5 было освоено связывание приложения на Python с базой данных под управлением PostgreSQL. Разработано приложение с графическим интерфейсом, полностью соответствующее требованиям методических указаний.

Приложение А1. Исходный код main.py

```
import sys
import json
from PyQt6.QtWidgets import (
    QApplication,
    QMainWindow,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QTableWidget,
    QTableWidgetItem,
    QPushButton,
    QLineEdit,
    QComboBox,
    QLabel,
    QDialog,
    QFormLayout,
    QDialogButtonBox,
    QMessageBox,
    QHeaderView,
    QAbstractItemView,
    QMenuBar,
    QStatusBar,
)
from PyQt6.QtGui import QAction
from PyQt6.QtCore import Qt, QTimer
from PyQt6.QtWidgets import QLineEdit
from database import Database


class SearchLineEdit(QLineEdit):
    """Поле поиска с поддержкой Esc для очистки"""

    def __init__(self, parent=None):
        super().__init__(parent)
        self.clear_callback = None
```

```
    def keyPressEvent(self, event):
        """Обработчик нажатий клавиш"""
        if event.key() == Qt.Key.Key_Escape:
            # Очищаем поле при нажатии Esc
            if self.clear_callback:
                self.clear_callback()
        else:
            # Обрабатываем остальные нажатия стандартно
            super().keyPressEvent(event)


class LoginDialog(QDialog):
    """Диалог входа в систему"""

    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db
        self.user_id = None
        self.setWindowTitle("Вход в систему")
        self.setModal(True)
        self.resize(350, 200)

        layout = QVBoxLayout(self)

        # Создаем форму
        form_layout = QFormLayout()

        self.username_edit = QLineEdit()
        self.username_edit.setPlaceholderText("Введите логин")
        self.password_edit = QLineEdit()
        self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)
        self.password_edit.setPlaceholderText("Введите пароль")

        form_layout.addRow("Логин:", self.username_edit)
        form_layout.addRow("Пароль:", self.password_edit)
```

```
        layout.addLayout(form_layout)

        # Кнопки
        buttons = QDialogButtonBox(
            QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel
        )
        buttons.accepted.connect(self.accept_login)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)

        # Кнопка регистрации
        self.register_button = QPushButton("Регистрация")
        self.register_button.clicked.connect(self.open_registration)
        layout.addWidget(self.register_button)

    def accept_login(self):
        """Обработка входа"""
        username = self.username_edit.text().strip()
        password = self.password_edit.text()

        if not username or not password:
            QMessageBox.warning(self, "Предупреждение", "Введите логин и пароль")
            return

        try:
            # Получаем хэш пароля из базы данных
            with self.db.conn.cursor() as cur:
                cur.execute(
                    "SELECT password_hash FROM users WHERE username = %s", (username,)
                )
                result = cur.fetchone()

            if result and (
                Database.verify_password(password, result[0]) or result[0] == password
            ):
                # Получаем данные пользователя
                with self.db.conn.cursor() as cur:
```

```
                    cur.execute(
                        "SELECT id, username, email FROM users WHERE username = %s",
                        (username,),
                    )
                    user_data = cur.fetchone()

                if user_data:
                    self.user_id = user_data[0]
                    self.accept()
                else:
                    QMessageBox.critical(self, "Ошибка", "Пользователь не найден")
            else:
                QMessageBox.critical(self, "Ошибка", "Неверный логин или пароль")

        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Ошибка входа: {str(e)}")

    def open_registration(self):
        """Открывает диалог регистрации"""
        dialog = RegisterDialog(db=self.db)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            # После успешной регистрации заполняем поля
            self.username_edit.setText(dialog.username_edit.text())


class RegisterDialog(QDialog):
    """Диалог регистрации"""

    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db
        self.setWindowTitle("Регистрация")
        self.setModal(True)
        self.resize(350, 250)

        layout = QVBoxLayout(self)
```

```
        # Создаем форму
        form_layout = QFormLayout()

        self.username_edit = QLineEdit()
        self.username_edit.setPlaceholderText("Введите логин")
        self.email_edit = QLineEdit()
        self.email_edit.setPlaceholderText("Введите email")
        self.password_edit = QLineEdit()
        self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)
        self.password_edit.setPlaceholderText("Введите пароль")
        self.confirm_password_edit = QLineEdit()
        self.confirm_password_edit.setEchoMode(QLineEdit.EchoMode.Password)
        self.confirm_password_edit.setPlaceholderText("Повторите пароль")

        form_layout.addRow("Логин:", self.username_edit)
        form_layout.addRow("Email:", self.email_edit)
        form_layout.addRow("Пароль:", self.password_edit)
        form_layout.addRow("Повтор пароля:", self.confirm_password_edit)

        layout.addLayout(form_layout)

        # Кнопки
        buttons = QDialogButtonBox(
            QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel
        )
        buttons.accepted.connect(self.accept_registration)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)

    def accept_registration(self):
        """Обработка регистрации"""
        username = self.username_edit.text().strip()
        email = self.email_edit.text().strip()
        password = self.password_edit.text()
        confirm_password = self.confirm_password_edit.text()

        # Валидация
```

```
        if not username or not email or not password:
            QMessageBox.warning(self, "Предупреждение", "Заполните все поля")
            return

        if password != confirm_password:
            QMessageBox.warning(self, "Предупреждение", "Пароли не совпадают")
            return

        if len(password) < 6:
            QMessageBox.warning(
                self, "Предупреждение", "Пароль должен быть не менее 6 символов"
            )
            return

        try:
            # Хэшируем пароль
            password_hash = Database.hash_password(password)

            # Создаем пользователя
            user_id = self.db.create_user(username, email, password_hash)

            QMessageBox.information(
                self, "Успех", f"Пользователь {username} успешно зарегистрирован!"
            )
            self.accept()

        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Ошибка регистрации: {str(e)}")


class DeviceDialog(QDialog):
    """Диалог для добавления/редактирования устройства"""

    def __init__(self, parent=None, device_id=None, db=None, user_id=None):
        super().__init__(parent)
        self.device_id = device_id
        self.db = db
```

```
        self.user_id = user_id
        self.setWindowTitle(
            "Добавить устройство" if device_id is None else "Редактировать устройство"
        )
        self.setModal(True)
        self.resize(400, 300)

        layout = QVBoxLayout(self)

        # Создаем форму
        form_layout = QFormLayout()

        # Поля формы
        self.name_edit = QLineEdit()
        form_layout.addRow("Название:", self.name_edit)

        self.hub_combo = QComboBox()
        self.type_combo = QComboBox()
        self.status_edit = QLineEdit()

        form_layout.addRow("Хаб:", self.hub_combo)
        form_layout.addRow("Тип устройства:", self.type_combo)
        form_layout.addRow("Статус (JSON):", self.status_edit)

        layout.addLayout(form_layout)

        # Кнопки
        buttons = QDialogButtonBox(
            QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel
        )
        buttons.accepted.connect(self.accept)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)

        # Загружаем данные
        self.load_data()
```

```
        # Если редактирование, загружаем данные устройства
        if device_id:
            self.load_device_data()

    def load_data(self):
        """Загружает хабы и типы устройств пользователя"""
        try:
            # Загружаем хабы пользователя
            hubs = self.db.get_user_hubs(self.user_id)
            self.hub_combo.clear()
            for hub_id, hub_name in hubs:
                self.hub_combo.addItem(hub_name, hub_id)

            # Загружаем типы устройств
            device_types = self.db.get_device_types()
            self.type_combo.clear()
            for type_id, type_name in device_types:
                self.type_combo.addItem(type_name, type_id)

        except Exception as e:
            QMessageBox.critical(
                self, "Ошибка", f"Не удалось загрузить данные: {str(e)}"
            )

    def load_device_data(self):
        """Загружает данные устройства для редактирования"""
        try:
            # Получаем данные устройства (простой запрос)
            with self.db.conn.cursor() as cur:
                cur.execute(
                    """
                    SELECT d.name, d.hub_id, d.type_id, d.status
                    FROM devices d
                    WHERE d.id = %s
                """,
                    (self.device_id,),
                )
```

```
                device = cur.fetchone()

            if device:
                self.name_edit.setText(device[0])
                self.hub_combo.setCurrentIndex(self.hub_combo.findData(device[1]))
                self.type_combo.setCurrentIndex(self.type_combo.findData(device[2]))
                self.status_edit.setText(json.dumps(device[3], ensure_ascii=False))

        except Exception as e:
            QMessageBox.critical(
                self, "Ошибка", f"Не удалось загрузить данные устройства: {str(e)}"
            )

    def get_data(self):
        """Возвращает данные из формы"""
        return {
            "name": self.name_edit.text().strip(),
            "hub_id": self.hub_combo.currentData(),
            "type_id": self.type_combo.currentData(),
            "status": self.status_edit.text().strip(),
        }


class MainWindow(QMainWindow):
    """Главное окно приложения"""

    def __init__(self):
        super().__init__()
        self.db = None
        self.current_user = None
        self.current_user_id = None

        # Показываем диалог входа перед инициализацией основного окна
        self.show_login_dialog()

    def show_login_dialog(self):
        """Показывает диалог входа"""
```

```
        login_dialog = LoginDialog(db=Database(), parent=self)
        if login_dialog.exec() == QDialog.DialogCode.Accepted:
            self.current_user_id = login_dialog.user_id
            self.initialize_main_window()
        else:
            sys.exit(0)  # Выход если пользователь отменил вход

    def initialize_main_window(self):
        """Инициализация главного окна после успешного входа"""
        self.setWindowTitle(
            f"Управление устройствами - Пользователь: {self.get_current_username()}"
        )
        self.setGeometry(100, 100, 1000, 600)

        # Создаем меню
        menubar = self.menuBar()

        user_menu = menubar.addMenu("Пользователь")
        logout_action = QAction("Выйти", self)
        logout_action.triggered.connect(self.logout)
        user_menu.addAction(logout_action)

        # Создаем центральный виджет
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        # Создаем layout
        layout = QVBoxLayout(central_widget)

        # Панель поиска и фильтров
        filter_layout = QHBoxLayout()

        self.filter_edit = SearchLineEdit()
        self.filter_edit.setPlaceholderText("Поиск по названию устройства...")
        self.filter_edit.returnPressed.connect(self.perform_search)  # Enter key
        self.filter_edit.clear_callback = (
            self.clear_search
```

```
)  # Connect clear callback for Esc key

self.clear_button = QPushButton("×")  # X button for clearing
self.clear_button.setMaximumWidth(30)
self.clear_button.setToolTip("Очистить поиск")
self.clear_button.clicked.connect(self.clear_search)
self.clear_button.setStyleSheet(
    """
    QPushButton {
        font-size: 16px;
        font-weight: bold;
        color: #666;
        border: none;
        background: transparent;
    }
    QPushButton:hover {
        color: #000;
        background: #f0f0f0;
    }
"""
)

self.search_button = QPushButton("Поиск")
self.search_button.clicked.connect(self.perform_search)
self.search_button.setMaximumWidth(100)

filter_layout.addWidget(QLabel("Фильтр:"))
filter_layout.addWidget(self.filter_edit, 1)

# Container for clear and search buttons
buttons_layout = QHBoxLayout()
buttons_layout.addWidget(self.clear_button)
buttons_layout.addWidget(self.search_button)

filter_layout.addLayout(buttons_layout)

layout.addLayout(filter_layout)
```

```
# Таблица устройств
self.table = QTableWidget()
self.table.setColumnCount(4)
self.table.setHorizontalHeaderLabels(["ID", "Название", "Тип", "Хаб"])
self.table.horizontalHeader().setSectionResizeMode(
    QHeaderView.ResizeMode.Stretch
)
self.table.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)
self.table.setEditTriggers(QAbstractItemView.EditTrigger.NoEditTriggers)

layout.addWidget(self.table)

# Панель кнопок
buttons_layout = QHBoxLayout()

self.add_button = QPushButton("Добавить устройство")
self.add_button.clicked.connect(self.add_device)

self.edit_button = QPushButton("Редактировать устройство")
self.edit_button.clicked.connect(self.edit_device)
self.edit_button.setEnabled(False)

self.delete_button = QPushButton("Удалить устройство")
self.delete_button.clicked.connect(self.delete_device)
self.delete_button.setEnabled(False)

self.refresh_button = QPushButton("Обновить")
self.refresh_button.clicked.connect(self.refresh_data)

buttons_layout.addWidget(self.add_button)
buttons_layout.addWidget(self.edit_button)
buttons_layout.addWidget(self.delete_button)
buttons_layout.addStretch()
buttons_layout.addWidget(self.refresh_button)

layout.addLayout(buttons_layout)
```

```
        # Статус бар
        self.statusBar().showMessage(f"Вошел как: {self.get_current_username()}")

        # Подключаемся к базе данных
        self.connect_to_database()

        # Загружаем данные
        self.refresh_data()

        # Подключаем сигналы
        self.table.itemSelectionChanged.connect(self.on_selection_changed)

    def get_current_username(self):
        """Получить имя текущего пользователя"""
        if self.current_user_id:
            try:
                with self.db.conn.cursor() as cur:
                    cur.execute(
                        "SELECT username FROM users WHERE id = %s",
                        (self.current_user_id,),
                    )
                    result = cur.fetchone()
                    return result[0] if result else "Неизвестный"
            except:
                pass
        return "Неизвестный"

    def connect_to_database(self):
        """Подключение к базе данных"""
        try:
            self.db = Database()
            self.statusBar().showMessage(
                f"Подключено к базе данных. Пользователь: {self.get_current_username()}"
            )
        except Exception as e:
            QMessageBox.critical(
```

```
                self,
                "Ошибка подключения",
                f"Не удалось подключиться к базе данных: {str(e)}",
            )
            self.statusBar().showMessage("Ошибка подключения к базе данных")

    def refresh_data(self):
        """Обновляет данные в таблице - показывает все устройства пользователя"""
        if not self.db or not self.current_user_id:
            return

        try:
            # Показываем все устройства пользователя без фильтрации
            devices = self.db.get_devices_for_user(self.current_user_id, "")

            self.table.setRowCount(len(devices))

            for row, device in enumerate(devices):
                for col, value in enumerate(device):
                    self.table.setItem(row, col, QTableWidgetItem(str(value)))

            device_count = self.db.get_user_devices_count(self.current_user_id)
            self.statusBar().showMessage(
                f"Устройств: {len(devices)} (всего: {device_count})"
            )

            # Очищаем поле поиска
            self.filter_edit.clear()

        except Exception as e:
            QMessageBox.critical(
                self, "Ошибка", f"Не удалось загрузить данные: {str(e)}"
            )
            self.statusBar().showMessage("Ошибка загрузки данных")

    def perform_search(self):
        """Выполняет поиск устройств"""
```

```
        if not self.db or not self.current_user_id:
            return

        try:
            filter_text = self.filter_edit.text().strip()
            devices = self.db.get_devices_for_user(self.current_user_id, filter_text)

            if not devices and filter_text:
                # Показываем ошибку если поиск не дал результатов
                QMessageBox.warning(
                    self, "Поиск", f"Устройства с названием '{filter_text}' не найдены"
                )
                return

            self.table.setRowCount(len(devices))

            for row, device in enumerate(devices):
                for col, value in enumerate(device):
                    self.table.setItem(row, col, QTableWidgetItem(str(value)))

            device_count = self.db.get_user_devices_count(self.current_user_id)
            self.statusBar().showMessage(
                f"Найдено устройств: {len(devices)} (всего: {device_count})"
            )

        except Exception as e:
            QMessageBox.critical(
                self, "Ошибка", f"Не удалось выполнить поиск: {str(e)}"
            )
            self.statusBar().showMessage("Ошибка поиска")

    def clear_search(self):
        """Очищает поле поиска и показывает все устройства"""
        self.filter_edit.clear()
        self.refresh_data()

    def on_selection_changed(self):
```

```
        """Обработчик изменения выделения в таблице"""
        selected = len(self.table.selectionModel().selectedRows()) > 0
        self.edit_button.setEnabled(selected)
        self.delete_button.setEnabled(selected)

    def add_device(self):
        """Добавляет новое устройство"""
        if not self.db or not self.current_user_id:
            return

        dialog = DeviceDialog(db=self.db, user_id=self.current_user_id)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            # Проверяем данные
            if not data["name"]:
                QMessageBox.warning(
                    self, "Предупреждение", "Название устройства обязательно"
                )
                return

            # Проверяем дубликат имени
            if self.db.device_exists(data["name"]):
                QMessageBox.warning(
                    self,
                    "Предупреждение",
                    f"Устройство с названием '{data['name']}' уже существует",
                )
                return

            try:
                # Парсим JSON статус
                status = json.loads(data["status"]) if data["status"] else {}

                # Сохраняем устройство
                device_id = self.db.save_device(
                    None,  # Новый device_id
```

```
                    data["hub_id"],
                    data["type_id"],
                    data["name"],
                    json.dumps(status, ensure_ascii=False),
                )

                self.refresh_data()
                self.statusBar().showMessage(f"Устройство добавлено с ID: {device_id}")

            except json.JSONDecodeError:
                QMessageBox.warning(
                    self, "Ошибка", "Неверный формат JSON в поле статуса"
                )
            except Exception as e:
                QMessageBox.critical(
                    self, "Ошибка", f"Не удалось добавить устройство: {str(e)}"
                )

    def edit_device(self):
        """Редактирует выбранное устройство"""
        if not self.db or not self.current_user_id:
            return

        current_row = self.table.currentRow()
        if current_row < 0:
            return

        device_id = int(self.table.item(current_row, 0).text())

        dialog = DeviceDialog(
            db=self.db, device_id=device_id, user_id=self.current_user_id
        )
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            # Проверяем данные
            if not data["name"]:
```

```
            QMessageBox.warning(
                self, "Предупреждение", "Название устройства обязательно"
            )
            return

        # Проверяем дубликат имени (исключая текущее устройство)
        if self.db.device_exists(data["name"], exclude_id=device_id):
            QMessageBox.warning(
                self,
                "Предупреждение",
                f"Устройство с названием '{data['name']}' уже существует",
            )
            return

        try:
            # Парсим JSON статус
            status = json.loads(data["status"]) if data["status"] else {}

            # Сохраняем устройство
            device_id = self.db.save_device(
                device_id,
                data["hub_id"],
                data["type_id"],
                data["name"],
                json.dumps(status, ensure_ascii=False),
            )

            self.refresh_data()
            self.statusBar().showMessage(f"Устройство обновлено с ID: {device_id}")

        except json.JSONDecodeError:
            QMessageBox.warning(
                self, "Ошибка", "Неверный формат JSON в поле статуса"
            )
        except Exception as e:
            QMessageBox.critical(
                self, "Ошибка", f"Не удалось обновить устройство: {str(e)}"
```

```
            )

    def delete_device(self):
        """Удаляет выбранное устройство"""
        if not self.db or not self.current_user_id:
            return

        current_row = self.table.currentRow()
        if current_row < 0:
            return

        device_id = int(self.table.item(current_row, 0).text())
        device_name = self.table.item(current_row, 1).text()

        reply = QMessageBox.question(
            self,
            "Подтверждение удаления",
            f"Вы действительно хотите удалить устройство '{device_name}'?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
            QMessageBox.StandardButton.No,
        )

        if reply == QMessageBox.StandardButton.Yes:
            try:
                # Используем безопасное удаление
                self.db.delete_device_safe(device_id)
                self.refresh_data()
                self.statusBar().showMessage(f"Устройство '{device_name}' удалено")

            except Exception as e:
                QMessageBox.critical(
                    self, "Ошибка", f"Не удалось удалить устройство: {str(e)}"
                )

    def logout(self):
        """Выход из системы"""
        reply = QMessageBox.question(
```

```
            self,
            "Подтверждение выхода",
            "Вы действительно хотите выйти?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
            QMessageBox.StandardButton.No,
        )

        if reply == QMessageBox.StandardButton.Yes:
            if self.db:
                self.db.close()
            self.close()
            # Перезапускаем приложение для повторного входа
            main()

    def closeEvent(self, event):
        """Обработчик закрытия окна"""
        if self.db:
            self.db.close()
        event.accept()


def main():
    """Главная функция"""
    app = QApplication(sys.argv)

    # Устанавливаем стиль
    app.setStyle("Fusion")

    # Создаем главное окно (вход будет показан автоматически)
    window = MainWindow()
    window.show()

    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```

Приложение А2. Исходный код database.py

```
import psycopg2
from psycopg2 import sql
import hashlib
import os


class Database:
    def __init__(self):
        self.conn = psycopg2.connect(
            host="localhost",
            port=5433,
            database="pozordom",
            user="pozordom_user",
            password="pozordom_pass",
        )
        self.conn.autocommit = True

    def close(self):
        self.conn.close()

    def get_hubs(self):
        """Получить список хабов: [(id, name), ...]"""
        with self.conn.cursor() as cur:
            cur.execute("SELECT id, name FROM hubs ORDER BY name")
            return cur.fetchall()

    def get_devices(self, filter_text=""):
        """Получить устройства с фильтрацией по имени"""
        query = """
            SELECT d.id, d.name, dt.type_name, h.name AS hub_name
            FROM devices d
            JOIN device_types dt ON d.type_id = dt.id
            JOIN hubs h ON d.hub_id = h.id
            WHERE d.name ILIKE %s
            ORDER BY d.id
```

```
        """
        with self.conn.cursor() as cur:
            cur.execute(query, (f"%{filter_text}%",))
            return cur.fetchall()

    def save_device(self, device_id, hub_id, type_id, name, status):
        """Вызов функции save_devices"""
        with self.conn.cursor() as cur:
            cur.callproc("save_devices", [device_id, hub_id, type_id, name, status])
            return cur.fetchone()[0]

    def delete_device(self, device_id):
        """Удаление напрямую (или через функцию, если реализована)"""
        with self.conn.cursor() as cur:
            cur.execute("DELETE FROM devices WHERE id = %s", (device_id,))

    def device_exists(self, name, exclude_id=None):
        """Проверка дубликата имени"""
        query = "SELECT 1 FROM devices WHERE name = %s"
        params = [name]
        if exclude_id:
            query += " AND id != %s"
            params.append(exclude_id)
        with self.conn.cursor() as cur:
            cur.execute(query, params)
            return cur.fetchone() is not None

    def get_user_by_credentials(self, username, password_hash):
        """Получить пользователя по логину и паролю"""
        with self.conn.cursor() as cur:
            cur.execute(
                """
                SELECT id, username, email FROM users
                WHERE username = %s AND password_hash = %s
            """,
                (username, password_hash),
            )
```

```
        return cur.fetchone()

    def create_user(self, username, email, password_hash):
        """Создать нового пользователя"""
        with self.conn.cursor() as cur:
            cur.execute(
                """
                INSERT INTO users (username, email, password_hash)
                VALUES (%s, %s, %s)
                RETURNING id
            """,
                (username, email, password_hash),
            )
            return cur.fetchone()[0]

    def get_user_hubs(self, user_id):
        """Получить хабы пользователя"""
        with self.conn.cursor() as cur:
            cur.execute(
                """
                SELECT id, name FROM hubs WHERE user_id = %s ORDER BY name
            """,
                (user_id,),
            )
            return cur.fetchall()

    def delete_device_safe(self, device_id):
        """Безопасное удаление устройства с очисткой логов"""
        with self.conn.cursor() as cur:
            # Сначала удаляем записи из log_devices
            cur.execute("DELETE FROM log_devices WHERE device_id = %s", (device_id,))
            # Затем удаляем само устройство
            cur.execute("DELETE FROM devices WHERE id = %s", (device_id,))

    def get_device_types(self):
        with self.conn.cursor() as cur:
            cur.execute("SELECT id, type_name FROM device_types ORDER BY type_name")
```

```
        return cur.fetchall()

@staticmethod
def hash_password(password):
    """Хэширование пароля с солью"""
    salt = os.urandom(32)
    pwdhash = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt, 100000)
    return salt + pwdhash

@staticmethod
def verify_password(password, stored_hash):
    """Проверка пароля"""
    try:
        salt = stored_hash[:32]
        stored_password_hash = stored_hash[32:]
        pwdhash = hashlib.pbkdf2_hmac(
            "sha256", password.encode("utf-8"), salt, 100000
        )
        return pwdhash == stored_password_hash
    except Exception:
        return False

def get_devices_for_user(self, user_id, filter_text=""):
    """Получить устройства пользователя с фильтрацией по имени"""
    query = """
        SELECT d.id, d.name, dt.type_name, h.name AS hub_name
        FROM devices d
        JOIN device_types dt ON d.type_id = dt.id
        JOIN hubs h ON d.hub_id = h.id
        WHERE h.user_id = %s AND d.name ILIKE %s
        ORDER BY d.id
    """
    with self.conn.cursor() as cur:
        cur.execute(query, (user_id, f"%{filter_text}%"))
        return cur.fetchall()

def get_user_devices_count(self, user_id):
```

```
"""Получить количество устройств пользователя"""
with self.conn.cursor() as cur:
    cur.execute(
        """
        SELECT COUNT(*) FROM devices d
        JOIN hubs h ON d.hub_id = h.id
        WHERE h.user_id = %s
    """,
        (user_id,),
    )
    return cur.fetchone()[0]
```