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

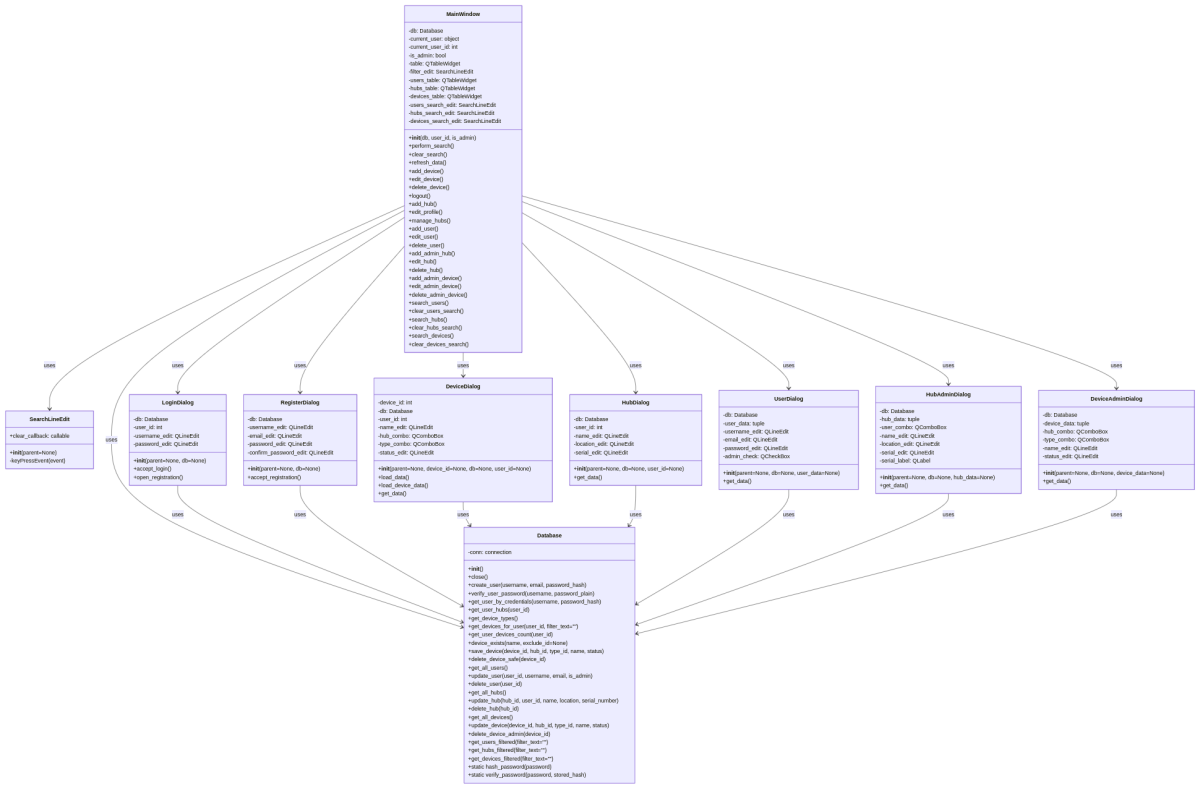

Рис. 1: Диаграмма классов приложения

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

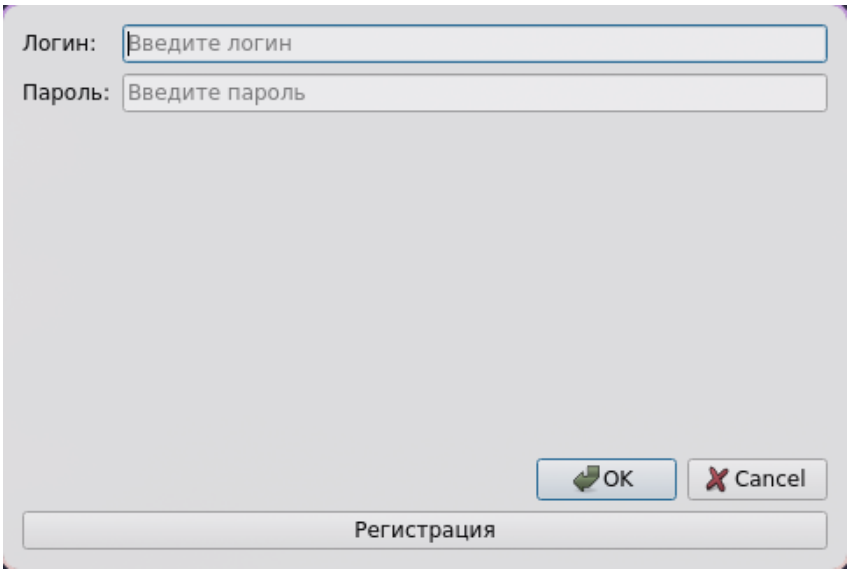

Рисунок 1 - Окно входа в систему

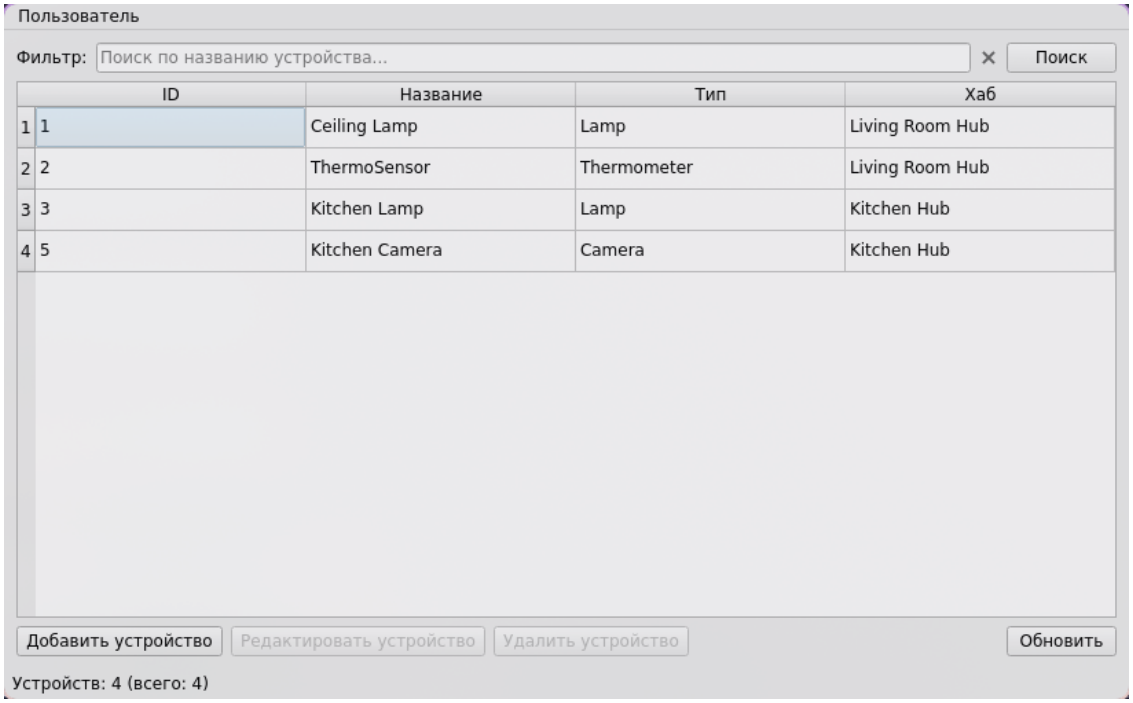

Рисунок 2 - Главное окно приложения

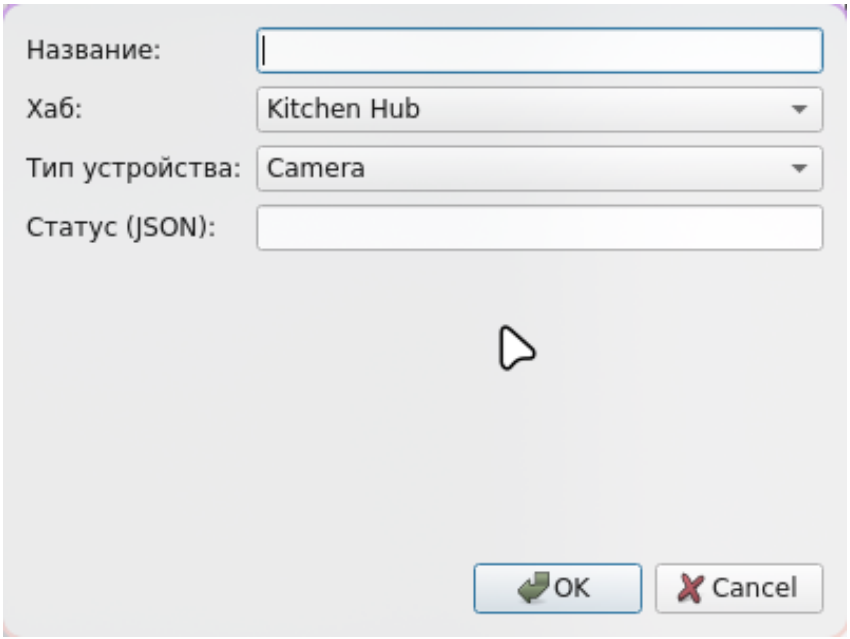

Рисунок 3 - Диалог добавления устройства

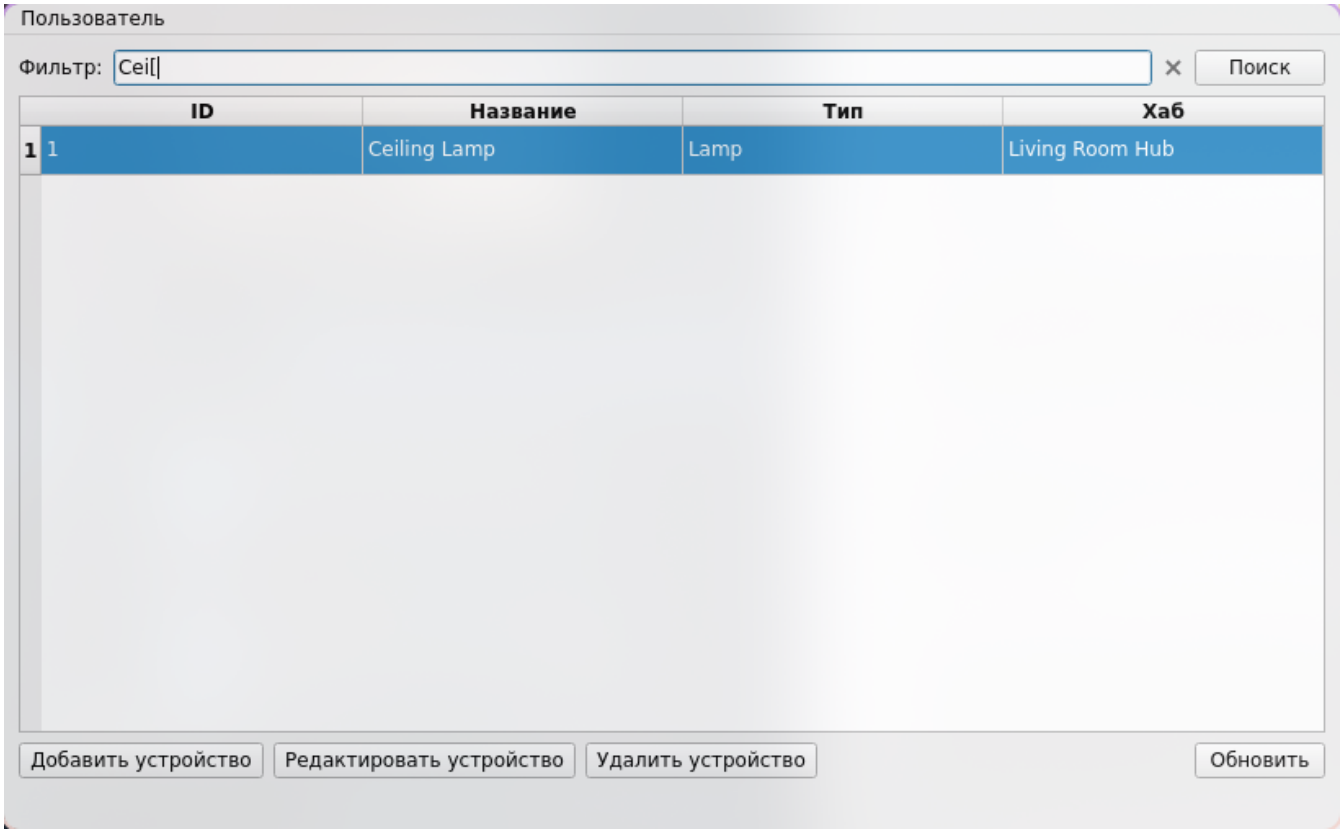

Рисунок 4 - Функционал ручного поиска

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,
    QTabWidget,
    QCheckBox,
)
from PyQt6.QtGui import QAction
from PyQt6.QtCore import Qt, QTimer
from database import Database
from widgets import SearchLineEdit
from dialogs import (
    LoginDialog,
    RegisterDialog,
    DeviceDialog,
    HubDialog,
    UserDialog,
```

```
    HubAdminDialog,
    DeviceAdminDialog,
)



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

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

        # 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.is_admin = login_dialog.is_admin
            self.initialize_main_window()
        else:
            sys.exit(0)  # Выход если пользователь отменил вход

    def initialize_main_window(self):
        """Инициализация главного окна после успешного входа"""
        if self.is_admin:
            # Admin panel
            self.setWindowTitle("Админ панель")
            self.setGeometry(100, 100, 1200, 700)

            menubar = self.menuBar()
```

```
user_menu = menubar.addMenu("Пользователь")
refresh_action = QAction("Обновить данные", self)
refresh_action.triggered.connect(self.refresh_admin_data)
logout_action = QAction("Выйти", self)
logout_action.triggered.connect(self.logout)
user_menu.addAction(refresh_action)
user_menu.addAction(logout_action)

tab_widget = QTabWidget(self)
self.setCentralWidget(tab_widget)

# Users tab
users_widget = QWidget()
tab_widget.addTab(users_widget, "Пользователи")
users_layout = QVBoxLayout(users_widget)

# Search for users
users_filter_layout = QHBoxLayout()

self.users_search_edit = SearchLineEdit()
self.users_search_edit.setPlaceholderText("Поиск по логину или email...")
self.users_search_edit.returnPressed.connect(self.search_users)
self.users_search_edit.clear_callback = self.clear_users_search

self.users_clear_button = QPushButton("×")
self.users_clear_button.setMaximumWidth(30)
self.users_clear_button.setToolTip("Очистить поиск")
self.users_clear_button.clicked.connect(self.clear_users_search)
self.users_clear_button.setStyleSheet(
    """
    QPushButton {
        font-size: 16px;
        font-weight: bold;
        color: #666;
        border: none;
        background: transparent;
    }
```

```
    QPushButton:hover {
        color: #000;
        background: #f0f0f0;
    }
"""
)

self.users_search_button = QPushButton("Поиск")
self.users_search_button.clicked.connect(self.search_users)
self.users_search_button.setMaximumWidth(100)

users_filter_layout.addWidget(QLabel("Фильтр:"))
users_filter_layout.addWidget(self.users_search_edit, 1)

# Container for clear and search buttons
users_buttons_layout = QHBoxLayout()
users_buttons_layout.addWidget(self.users_clear_button)
users_buttons_layout.addWidget(self.users_search_button)

users_filter_layout.addLayout(users_buttons_layout)

users_layout.addLayout(users_filter_layout)

self.users_table = QTableWidget()
self.users_table.setColumnCount(4)
self.users_table.setHorizontalHeaderLabels(["ID", "Логин", "Email", "Админ"])
self.users_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode
users_layout.addWidget(self.users_table)
users_btns = QHBoxLayout()
add_user_btn = QPushButton("Добавить пользователя")
add_user_btn.clicked.connect(self.add_user)
edit_user_btn = QPushButton("Редактировать")
edit_user_btn.clicked.connect(self.edit_user)
delete_user_btn = QPushButton("Удалить")
delete_user_btn.clicked.connect(self.delete_user)
refresh_users_btn = QPushButton("Обновить")
refresh_users_btn.clicked.connect(self.refresh_users_table)
```

```
users_btns.addWidget(add_user_btn)
users_btns.addWidget(edit_user_btn)
users_btns.addWidget(delete_user_btn)
users_btns.addWidget(refresh_users_btn)
users_layout.addLayout(users_btns)

# Hubs tab
hubs_widget = QWidget()
tab_widget.addTab(hubs_widget, "Хабы")
hubs_layout = QVBoxLayout(hubs_widget)

# Search for hubs
hubs_filter_layout = QHBoxLayout()

self.hubs_search_edit = SearchLineEdit()
self.hubs_search_edit.setPlaceholderText("Поиск по названию, местоположению, се
self.hubs_search_edit.returnPressed.connect(self.search_hubs)
self.hubs_search_edit.clear_callback = self.clear_hubs_search

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

```
)

self.hubs_search_button = QPushButton("Поиск")
self.hubs_search_button.clicked.connect(self.search_hubs)
self.hubs_search_button.setMaximumWidth(100)

hubs_filter_layout.addWidget(QLabel("Фильтр:"))
hubs_filter_layout.addWidget(self.hubs_search_edit, 1)

# Container for clear and search buttons
hubs_buttons_layout = QHBoxLayout()
hubs_buttons_layout.addWidget(self.hubs_clear_button)
hubs_buttons_layout.addWidget(self.hubs_search_button)

hubs_filter_layout.addLayout(hubs_buttons_layout)

hubs_layout.addLayout(hubs_filter_layout)

self.hubs_table = QTableWidget()
self.hubs_table.setColumnCount(5)
self.hubs_table.setHorizontalHeaderLabels(["ID", "Владелец", "Название", "Мест
self.hubs_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode
hubs_layout.addWidget(self.hubs_table)
hubs_btns = QHBoxLayout()
add_hub_btn = QPushButton("Добавить хаб")
add_hub_btn.clicked.connect(self.add_admin_hub)
edit_hub_btn = QPushButton("Редактировать")
edit_hub_btn.clicked.connect(self.edit_hub)
delete_hub_btn = QPushButton("Удалить")
delete_hub_btn.clicked.connect(self.delete_hub)
refresh_hubs_btn = QPushButton("Обновить")
refresh_hubs_btn.clicked.connect(self.refresh_hubs_table)
hubs_btns.addWidget(add_hub_btn)
hubs_btns.addWidget(edit_hub_btn)
hubs_btns.addWidget(delete_hub_btn)
hubs_btns.addWidget(refresh_hubs_btn)
hubs_layout.addLayout(hubs_btns)
```

```
# Devices tab
devices_widget = QWidget()
tab_widget.addTab(devices_widget, "Устройства")
devices_layout = QVBoxLayout(devices_widget)

# Search for devices
devices_filter_layout = QHBoxLayout()

self.devices_search_edit = SearchLineEdit()
self.devices_search_edit.setPlaceholderText("Поиск по названию, типу или хабу.
self.devices_search_edit.returnPressed.connect(self.search_devices)
self.devices_search_edit.clear_callback = self.clear_devices_search

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

self.devices_search_button = QPushButton("Поиск")
self.devices_search_button.clicked.connect(self.search_devices)
self.devices_search_button.setMaximumWidth(100)
```

```
        devices_filter_layout.addWidget(QLabel("Фильтр:"))
        devices_filter_layout.addWidget(self.devices_search_edit, 1)

        # Container for clear and search buttons
        devices_buttons_layout = QHBoxLayout()
        devices_buttons_layout.addWidget(self.devices_clear_button)
        devices_buttons_layout.addWidget(self.devices_search_button)

        devices_filter_layout.addLayout(devices_buttons_layout)

        devices_layout.addLayout(devices_filter_layout)

        self.devices_table = QTableWidget()
        self.devices_table.setColumnCount(5)
        self.devices_table.setHorizontalHeaderLabels(["ID", "Хаб", "Название", "Тип", "
        self.devices_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMo
        devices_layout.addWidget(self.devices_table)
        devices_btns = QHBoxLayout()
        add_device_btn = QPushButton("Добавить устройство")
        add_device_btn.clicked.connect(self.add_admin_device)
        edit_device_btn = QPushButton("Редактировать")
        edit_device_btn.clicked.connect(self.edit_admin_device)
        delete_device_btn = QPushButton("Удалить")
        delete_device_btn.clicked.connect(self.delete_admin_device)
        refresh_devices_btn = QPushButton("Обновить")
        refresh_devices_btn.clicked.connect(self.refresh_devices_table)
        devices_btns.addWidget(add_device_btn)
        devices_btns.addWidget(edit_device_btn)
        devices_btns.addWidget(delete_device_btn)
        devices_btns.addWidget(refresh_devices_btn)
        devices_layout.addLayout(devices_btns)

        self.connect_to_database()
        self.refresh_admin_data()
    else:
        # Regular user UI
```

```
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")
profile_action = QAction("Профиль", self)
profile_action.triggered.connect(self.edit_profile)
logout_action = QAction("Выйти", self)
logout_action.triggered.connect(self.logout)
user_menu.addAction(profile_action)
user_menu.addAction(logout_action)

hub_menu = menubar.addMenu("Хабы")
add_hub_action = QAction("Добавить хаб", self)
add_hub_action.triggered.connect(self.add_hub)
manage_hubs_action = QAction("Управление хабами", self)
manage_hubs_action.triggered.connect(self.manage_hubs)
hub_menu.addAction(add_hub_action)
hub_menu.addAction(manage_hubs_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 add_hub(self):
        """Добавляет новый хаб"""
        if not self.db or not self.current_user_id:
            return

        dialog = HubDialog(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"] or not data["location"] or not data["serial_number"]:
                QMessageBox.warning(
                    self, "Предупреждение", "Заполните все поля"
                )
                return

            try:
                # Добавляем хаб
                with self.db.conn.cursor() as cur:
                    cur.execute(
                        "INSERT INTO hubs (user_id, name, location, serial_number) VALUES
                        (self.current_user_id, data['name'], data['location'], data['serial
```

```
                )
                hub_id = cur.fetchone()[0]
            self.db.conn.commit()

            QMessageBox.information(self, "Успех", f"Хаб '{data['name']}' добавлен с ID

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

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()
        if not self.is_admin:  # Only show status for regular users
            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)}",
        )
        if not self.is_admin:  # Only show status for regular users
            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()
            # Сбрасываем состояние и показываем диалог входа снова
            self.reset_and_show_login()

    def reset_and_show_login(self):
        """Сбрасывает состояние окна и показывает диалог входа"""
        # Очищаем центральный виджет
        central_widget = self.centralWidget()
        if central_widget:
            central_widget.setParent(None)
            central_widget.deleteLater()

        # Очищаем меню
        menubar = self.menuBar()
        if menubar:
            menubar.clear()

        # Сбрасываем статус бар
        status_bar = self.statusBar()
        if status_bar:
            status_bar.clearMessage()

        # Сбрасываем состояние
        self.db = None
        self.current_user = None
        self.current_user_id = None
        self.is_admin = False
```

```
        # Убираем все атрибуты таблиц и других виджетов
        for attr in ['users_table', 'hubs_table', 'devices_table', 'table',
                     'filter_edit', 'clear_button', 'search_button',
                     'add_button', 'edit_button', 'delete_button', 'refresh_button',
                     'users_search_edit', 'hubs_search_edit', 'devices_search_edit']:
            if hasattr(self, attr):
                delattr(self, attr)

        # Показываем диалог входа снова
        self.show_login_dialog()

    def edit_profile(self):
        """Редактирование профиля пользователя"""
        if not self.db or not self.current_user_id:
            return

        # Get current user data
        try:
            with self.db.conn.cursor() as cur:
                cur.execute("SELECT username, email FROM users WHERE id = %s", (self.curre
                user_data = cur.fetchone()
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось загрузить данные профиля: {st
            return

        if not user_data:
            QMessageBox.critical(self, "Ошибка", "Пользователь не найден")
            return

        # Dialog for editing profile
        dialog = QDialog(self)
        dialog.setWindowTitle("Редактирование профиля")
        dialog.setModal(True)
        dialog.resize(350, 200)

        layout = QVBoxLayout(dialog)
        form = QFormLayout()
```

```
username_edit = QLineEdit(user_data[0])
email_edit = QLineEdit(user_data[1])
password_edit = QLineEdit()
password_edit.setEchoMode(QLineEdit.EchoMode.Password)
password_edit.setPlaceholderText("Оставьте пустым, чтобы не менять")

form.addRow("Логин:", username_edit)
form.addRow("Email:", email_edit)
form.addRow("Новый пароль:", password_edit)
layout.addLayout(form)

buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.St
buttons.accepted.connect(dialog.accept)
buttons.rejected.connect(dialog.reject)
layout.addWidget(buttons)

if dialog.exec() == QDialog.DialogCode.Accepted:
    username = username_edit.text().strip()
    email = email_edit.text().strip()
    password = password_edit.text()

    if not username or not email:
        QMessageBox.warning(self, "Ошибка", "Логин и email обязательны")
        return

    try:
        # Update user
        if password:
            password_hash = Database.hash_password(password)
            with self.db.conn.cursor() as cur:
                cur.execute(
                    "UPDATE users SET username = %s, email = %s, password_hash = %s
                    (username, email, password_hash, self.current_user_id)
                )
        else:
            with self.db.conn.cursor() as cur:
```

```
                    cur.execute(
                        "UPDATE users SET username = %s, email = %s WHERE id = %s",
                        (username, email, self.current_user_id)
                    )
                self.db.conn.commit()
                QMessageBox.information(self, "Успех", "Профиль обновлен")
                self.setWindowTitle(f"Управление устройствами - Пользователь: {username}")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось обновить профиль: {str(e)

    def manage_hubs(self):
        """Управление хабами пользователя"""
        if not self.db or not self.current_user_id:
            return

        # Dialog with table of user's hubs
        dialog = QDialog(self)
        dialog.setWindowTitle("Управление хабами")
        dialog.setModal(True)
        dialog.resize(600, 400)

        layout = QVBoxLayout(dialog)

        table = QTableWidget()
        table.setColumnCount(4)
        table.setHorizontalHeaderLabels(["ID", "Название", "Местоположение", "Серийный номе
        table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

        # Load user's hubs with full data
        try:
            with self.db.conn.cursor() as cur:
                cur.execute("SELECT id, name, location, serial_number FROM hubs WHERE user_
                hubs = cur.fetchall()
            table.setRowCount(len(hubs))
            for row, (hub_id, hub_name, location, serial) in enumerate(hubs):
                table.setItem(row, 0, QTableWidgetItem(str(hub_id)))
                table.setItem(row, 1, QTableWidgetItem(hub_name))
```

```
                table.setItem(row, 2, QTableWidgetItem(location or ""))
                table.setItem(row, 3, QTableWidgetItem(serial or ""))
        except Exception as e:
            QMessageBox.critical(dialog, "Ошибка", f"Не удалось загрузить хабы: {str(e)}")
            return

        layout.addWidget(table)

        buttons = QHBoxLayout()
        edit_btn = QPushButton("Редактировать")
        edit_btn.clicked.connect(lambda: self.edit_user_hub(table, dialog))
        delete_btn = QPushButton("Удалить")
        delete_btn.clicked.connect(lambda: self.delete_user_hub(table, dialog))
        close_btn = QPushButton("Закрыть")
        close_btn.clicked.connect(dialog.accept)

        buttons.addWidget(edit_btn)
        buttons.addWidget(delete_btn)
        buttons.addStretch()
        buttons.addWidget(close_btn)
        layout.addLayout(buttons)

        dialog.exec()

    def edit_user_hub(self, table, parent_dialog):
        """Edit selected hub"""
        row = table.currentRow()
        if row < 0:
            QMessageBox.warning(parent_dialog, "Ошибка", "Выберите хаб")
            return

        hub_id = int(table.item(row, 0).text())
        name = table.item(row, 1).text()

        # Simple edit dialog
        dialog = QDialog(parent_dialog)
        dialog.setWindowTitle("Редактировать хаб")
```

```
dialog.setModal(True)
dialog.resize(350, 150)

layout = QVBoxLayout(dialog)
form = QFormLayout()

name_edit = QLineEdit(name)
location_edit = QLineEdit()
# Показываем серийный номер как статический текст
serial_label = QLabel()

# Загружаем текущий серийный номер
try:
    with self.db.conn.cursor() as cur:
        cur.execute("SELECT serial_number FROM hubs WHERE id = %s", (hub_id,))
        current_serial = cur.fetchone()
        if current_serial:
            serial_label.setText(current_serial[0])
except:
    serial_label.setText("Неизвестен")

form.addRow("Название:", name_edit)
form.addRow("Местоположение:", location_edit)
form.addRow("Серийный номер:", serial_label)
layout.addLayout(form)

buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.St
buttons.accepted.connect(dialog.accept)
buttons.rejected.connect(dialog.reject)
layout.addWidget(buttons)

if dialog.exec() == QDialog.DialogCode.Accepted:
    new_name = name_edit.text().strip()
    location = location_edit.text().strip()
    # Используем текущий серийный номер из базы данных
    serial = serial_label.text()
```

```
            if not new_name:
                QMessageBox.warning(dialog, "Ошибка", "Название обязательно")
                return

            try:
                with self.db.conn.cursor() as cur:
                    cur.execute(
                        "UPDATE hubs SET name = %s, location = %s, serial_number = %s WHERE
                        (new_name, location, serial, hub_id, self.current_user_id)
                    )
                self.db.conn.commit()
                QMessageBox.information(dialog, "Успех", "Хаб обновлен")
                # Refresh table
                table.item(row, 1).setText(new_name)
                table.item(row, 2).setText(location)
                table.item(row, 3).setText(serial)
            except Exception as e:
                QMessageBox.critical(dialog, "Ошибка", f"Не удалось обновить хаб: {str(e)}"

    def delete_user_hub(self, table, parent_dialog):
        """Delete selected hub"""
        row = table.currentRow()
        if row < 0:
            QMessageBox.warning(parent_dialog, "Ошибка", "Выберите хаб")
            return

        hub_id = int(table.item(row, 0).text())
        name = table.item(row, 1).text()

        reply = QMessageBox.question(
            parent_dialog, "Подтверждение",
            f"Удалить хаб '{name}'? Все устройства в этом хабе также будут удалены.",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
        )

        if reply == QMessageBox.StandardButton.Yes:
            try:
```

```
                with self.db.conn.cursor() as cur:
                    cur.execute("DELETE FROM hubs WHERE id = %s AND user_id = %s", (hub_id
                self.db.conn.commit()
                QMessageBox.information(parent_dialog, "Успех", "Хаб удален")
                table.removeRow(row)
            except Exception as e:
                QMessageBox.critical(parent_dialog, "Ошибка", f"Не удалось удалить хаб: {st

    # Admin methods
    def refresh_admin_data(self):
        """Refresh all admin tables"""
        try:
            # Users
            users = self.db.get_all_users()
            self.users_table.setRowCount(0)  # Clear table first
            self.users_table.setRowCount(len(users))
            for row, (user_id, username, email, is_admin) in enumerate(users):
                self.users_table.setItem(row, 0, QTableWidgetItem(str(user_id)))
                self.users_table.setItem(row, 1, QTableWidgetItem(username))
                self.users_table.setItem(row, 2, QTableWidgetItem(email))
                self.users_table.setItem(row, 3, QTableWidgetItem("Да" if is_admin else "Не

            # Hubs
            hubs = self.db.get_all_hubs()
            self.hubs_table.setRowCount(0)  # Clear table first
            self.hubs_table.setRowCount(len(hubs))
            for row, (hub_id, user_id, username, name, location, serial) in enumerate(hubs)
                self.hubs_table.setItem(row, 0, QTableWidgetItem(str(hub_id)))
                self.hubs_table.setItem(row, 1, QTableWidgetItem(username))
                self.hubs_table.setItem(row, 2, QTableWidgetItem(name))
                self.hubs_table.setItem(row, 3, QTableWidgetItem(location))
                self.hubs_table.setItem(row, 4, QTableWidgetItem(serial))

            # Devices
            devices = self.db.get_all_devices()
            self.devices_table.setRowCount(0)  # Clear table first
            for device in devices:
```

```
                row = self.devices_table.rowCount()
                self.devices_table.insertRow(row)
                device_id, hub_id, hub_name, name, type_name, status = device
                self.devices_table.setItem(row, 0, QTableWidgetItem(str(device_id)))
                self.devices_table.setItem(row, 1, QTableWidgetItem(str(hub_name) if hub_na
                self.devices_table.setItem(row, 2, QTableWidgetItem(str(name) if name else
                self.devices_table.setItem(row, 3, QTableWidgetItem(str(type_name) if type_
                self.devices_table.setItem(row, 4, QTableWidgetItem(str(status) if status e

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

    def add_user(self):
        """Add new user"""
        dialog = UserDialog(db=self.db)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            if not data["username"] or not data["email"] or not data["password"]:
                QMessageBox.warning(self, "Ошибка", "Заполните все поля")
                return

            try:
                password_hash = Database.hash_password(data["password"])
                self.db.create_user(data["username"], data["email"], password_hash)
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Пользователь добавлен")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось добавить пользователя: {s

    def edit_user(self):
        """Edit selected user"""
        row = self.users_table.currentRow()
        if row < 0:
            QMessageBox.warning(self, "Ошибка", "Выберите пользователя")
            return
```

```
user_id = int(self.users_table.item(row, 0).text())
old_username = self.users_table.item(row, 1).text()
old_email = self.users_table.item(row, 2).text()
old_admin = self.users_table.item(row, 3).text() == "Да"

dialog = QDialog(self)
dialog.setWindowTitle("Редактировать пользователя")
dialog.setModal(True)
dialog.resize(350, 150)

layout = QVBoxLayout(dialog)
form = QFormLayout()
username_edit = QLineEdit(old_username)
email_edit = QLineEdit(old_email)
admin_check = QCheckBox("Администратор")
admin_check.setChecked(old_admin)

form.addRow("Логин:", username_edit)
form.addRow("Email:", email_edit)
form.addRow(admin_check)
layout.addLayout(form)

buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.St
buttons.accepted.connect(dialog.accept)
buttons.rejected.connect(dialog.reject)
layout.addWidget(buttons)

if dialog.exec() == QDialog.DialogCode.Accepted:
    username = username_edit.text().strip()
    email = email_edit.text().trimmed()
    is_admin = admin_check.isChecked()

    if not username or not email:
        QMessageBox.warning(dialog, "Ошибка", "Заполните все поля")
        return

    try:
```

```
                self.db.update_user(user_id, username, email, is_admin)
                self.refresh_admin_data()
                QMessageBox.information(dialog, "Успех", "Пользователь обновлен")
            except Exception as e:
                QMessageBox.critical(dialog, "Ошибка", f"Не удалось обновить пользователя:

    def delete_user(self):
        """Delete selected user"""
        row = self.users_table.currentRow()
        if row < 0:
            QMessageBox.warning(self, "Ошибка", "Выберите пользователя")
            return

        user_id = int(self.users_table.item(row, 0).text())
        username = self.users_table.item(row, 1).text()

        reply = QMessageBox.question(self, "Подтверждение", f"Удалить пользователя '{userna
        if reply == QMessageBox.StandardButton.Yes:
            try:
                self.db.delete_user(user_id)
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Пользователь удален")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось удалить пользователя: {st

    def add_admin_hub(self):
        """Add hub as admin"""
        dialog = HubAdminDialog(db=self.db)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            if not data["name"] or not data["location"] or not data["serial_number"]:
                QMessageBox.warning(self, "Ошибка", "Заполните все поля")
                return

            try:
                self.db.create_hub(data["user_id"], data["name"], data["location"], data["
```

```
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Хаб добавлен")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось добавить хаб: {str(e)}")

    def edit_hub(self):
        """Edit selected hub"""
        row = self.hubs_table.currentRow()
        if row < 0:
            QMessageBox.warning(self, "Ошибка", "Выберите хаб")
            return

        hub_id = int(self.hubs_table.item(row, 0).text())
        old_username = self.hubs_table.item(row, 1).text()
        old_name = self.hubs_table.item(row, 2).text()
        old_location = self.hubs_table.item(row, 3).text()
        old_serial = self.hubs_table.item(row, 4).text()

        # Get user_id from username
        users = self.db.get_all_users()
        old_user_id = None
        for uid, uname, email, is_admin in users:
            if uname == old_username:
                old_user_id = uid
                break

        if old_user_id is None:
            QMessageBox.critical(self, "Ошибка", "Не удалось найти пользователя")
            return

        dialog = HubAdminDialog(db=self.db, hub_data=(hub_id, old_user_id, old_username, ol
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            if not data["name"] or not data["location"] or not data["serial_number"]:
                QMessageBox.warning(self, "Ошибка", "Заполните все поля")
                return
```

```
            try:
                self.db.update_hub(hub_id, data["user_id"], data["name"], data["location"],
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Хаб обновлен")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось обновить хаб: {str(e)}")

    def delete_hub(self):
        """Delete selected hub"""
        row = self.hubs_table.currentRow()
        if row < 0:
            QMessageBox.warning(self, "Ошибка", "Выберите хаб")
            return

        hub_id = int(self.hubs_table.item(row, 0).text())
        name = self.hubs_table.item(row, 2).text()

        reply = QMessageBox.question(self, "Подтверждение", f"Удалить хаб '{name}'?")
        if reply == QMessageBox.StandardButton.Yes:
            try:
                self.db.delete_hub(hub_id)
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Хаб удален")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось удалить хаб: {str(e)}")

    def add_admin_device(self):
        """Add device as admin"""
        dialog = DeviceAdminDialog(db=self.db)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            data = dialog.get_data()

            if not data["name"]:
                QMessageBox.warning(self, "Ошибка", "Введите название")
                return
```

```
            try:
                self.db.save_device(None, data["hub_id"], data["type_id"], data["name"], da
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Устройство добавлено")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось добавить устройство: {str

    def edit_admin_device(self):
        """Edit device as admin"""
        row = self.devices_table.currentRow()
        if row < 0:
            QMessageBox.warning(self, "Ошибка", "Выберите устройство")
            return

        item = self.devices_table.item(row, 0)
        if item is None:
            QMessageBox.warning(self, "Ошибка", "Выберите устройство с данными")
            return

        device_id = int(item.text())
        hub_name = self.devices_table.item(row, 1).text()
        name = self.devices_table.item(row, 2).text()
        type_name = self.devices_table.item(row, 3).text()
        status = self.devices_table.item(row, 4).text()

        # Get hub_id from hub_name
        try:
            with self.db.conn.cursor() as cur:
                cur.execute("SELECT id FROM hubs WHERE name = %s", (hub_name,))
                hub_result = cur.fetchone()
                hub_id = hub_result[0] if hub_result else None
        except:
            hub_id = None

        # Get type_id from type_name
        try:
            with self.db.conn.cursor() as cur:
```

```
                cur.execute("SELECT id FROM device_types WHERE type_name = %s", (type_name
                type_result = cur.fetchone()
                type_id = type_result[0] if type_result else None
            except:
                type_id = None

            if hub_id is None or type_id is None:
                QMessageBox.critical(self, "Ошибка", "Не удалось найти хаб или тип")
                return

            dialog = DeviceAdminDialog(db=self.db, device_data=(device_id, hub_id, hub_name, na
            if dialog.exec() == QDialog.DialogCode.Accepted:
                data = dialog.get_data()

                if not data["name"]:
                    QMessageBox.warning(self, "Ошибка", "Введите название")
                    return

                try:
                    self.db.update_device(device_id, data["hub_id"], data["type_id"], data["nam
                    self.refresh_admin_data()
                    QMessageBox.information(self, "Успех", "Устройство обновлено")
                except Exception as e:
                    QMessageBox.critical(self, "Ошибка", f"Не удалось обновить устройство: {str

        def delete_admin_device(self):
            """Delete device as admin"""
            row = self.devices_table.currentRow()
            if row < 0:
                QMessageBox.warning(self, "Ошибка", "Выберите устройство")
                return

            device_id = int(self.devices_table.item(row, 0).text())
            name = self.devices_table.item(row, 2).text()

            reply = QMessageBox.question(self, "Подтверждение", f"Удалить устройство '{name}'?"
            if reply == QMessageBox.StandardButton.Yes:
```

```
            try:
                self.db.delete_device_admin(device_id)
                self.refresh_admin_data()
                QMessageBox.information(self, "Успех", "Устройство удалено")
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", f"Не удалось удалить устройство: {str

    def refresh_users_table(self):
        """Refresh only the users table"""
        try:
            users = self.db.get_all_users()
            self.users_table.setRowCount(0)
            self.users_table.setRowCount(len(users))
            for row, (user_id, username, email, is_admin) in enumerate(users):
                self.users_table.setItem(row, 0, QTableWidgetItem(str(user_id)))
                self.users_table.setItem(row, 1, QTableWidgetItem(username))
                self.users_table.setItem(row, 2, QTableWidgetItem(email))
                self.users_table.setItem(row, 3, QTableWidgetItem("Да" if is_admin else "Не
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось обновить таблицу пользовател

    def refresh_hubs_table(self):
        """Refresh only the hubs table"""
        try:
            hubs = self.db.get_all_hubs()
            self.hubs_table.setRowCount(0)
            self.hubs_table.setRowCount(len(hubs))
            for row, (hub_id, user_id, username, name, location, serial) in enumerate(hubs
                self.hubs_table.setItem(row, 0, QTableWidgetItem(str(hub_id)))
                self.hubs_table.setItem(row, 1, QTableWidgetItem(username))
                self.hubs_table.setItem(row, 2, QTableWidgetItem(name))
                self.hubs_table.setItem(row, 3, QTableWidgetItem(location))
                self.hubs_table.setItem(row, 4, QTableWidgetItem(serial))
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось обновить таблицу хабов: {str

    def refresh_devices_table(self):
```

```
        """Refresh only the devices table"""
        try:
            devices = self.db.get_all_devices()
            self.devices_table.setRowCount(0)
            for device in devices:
                row = self.devices_table.rowCount()
                self.devices_table.insertRow(row)
                device_id, hub_id, hub_name, name, type_name, status = device
                self.devices_table.setItem(row, 0, QTableWidgetItem(str(device_id)))
                self.devices_table.setItem(row, 1, QTableWidgetItem(str(hub_name) if hub_na
                self.devices_table.setItem(row, 2, QTableWidgetItem(str(name) if name else
                self.devices_table.setItem(row, 3, QTableWidgetItem(str(type_name) if type_
                self.devices_table.setItem(row, 4, QTableWidgetItem(str(status) if status e
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось обновить таблицу устройств: +

    # Search methods for admin panel
    def search_users(self):
        """Search users in admin panel"""
        try:
            filter_text = self.users_search_edit.text().strip()
            if filter_text:
                users = self.db.get_users_filtered(filter_text)
            else:
                users = self.db.get_all_users()

            self.users_table.setRowCount(0)
            self.users_table.setRowCount(len(users))
            for row, (user_id, username, email, is_admin) in enumerate(users):
                self.users_table.setItem(row, 0, QTableWidgetItem(str(user_id)))
                self.users_table.setItem(row, 1, QTableWidgetItem(username))
                self.users_table.setItem(row, 2, QTableWidgetItem(email))
                self.users_table.setItem(row, 3, QTableWidgetItem("Да" if is_admin else "Не
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось выполнить поиск пользователей

    def clear_users_search(self):
```

```
        """Clear users search and show all users"""
        self.users_search_edit.clear()
        self.search_users()

    def search_hubs(self):
        """Search hubs in admin panel"""
        try:
            filter_text = self.hubs_search_edit.text().strip()
            if filter_text:
                hubs = self.db.get_hubs_filtered(filter_text)
            else:
                hubs = self.db.get_all_hubs()

            self.hubs_table.setRowCount(0)
            self.hubs_table.setRowCount(len(hubs))
            for row, (hub_id, user_id, username, name, location, serial) in enumerate(hubs
                self.hubs_table.setItem(row, 0, QTableWidgetItem(str(hub_id)))
                self.hubs_table.setItem(row, 1, QTableWidgetItem(username))
                self.hubs_table.setItem(row, 2, QTableWidgetItem(name))
                self.hubs_table.setItem(row, 3, QTableWidgetItem(location))
                self.hubs_table.setItem(row, 4, QTableWidgetItem(serial))
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось выполнить поиск хабов: {str(e

    def clear_hubs_search(self):
        """Clear hubs search and show all hubs"""
        self.hubs_search_edit.clear()
        self.search_hubs()

    def search_devices(self):
        """Search devices in admin panel"""
        try:
            filter_text = self.devices_search_edit.text().strip()
            if filter_text:
                devices = self.db.get_devices_filtered(filter_text)
            else:
                devices = self.db.get_all_devices()
```

```
            self.devices_table.setRowCount(0)
            for device in devices:
                row = self.devices_table.rowCount()
                self.devices_table.insertRow(row)
                device_id, hub_id, hub_name, name, type_name, status = device
                self.devices_table.setItem(row, 0, QTableWidgetItem(str(device_id)))
                self.devices_table.setItem(row, 1, QTableWidgetItem(str(hub_name) if hub_na
                self.devices_table.setItem(row, 2, QTableWidgetItem(str(name) if name else
                self.devices_table.setItem(row, 3, QTableWidgetItem(str(type_name) if type_
                self.devices_table.setItem(row, 4, QTableWidgetItem(str(status) if status e
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось выполнить поиск устройств: {s

    def clear_devices_search(self):
        """Clear devices search and show all devices"""
        self.devices_search_edit.clear()
        self.search_devices()

    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, is_admin 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]

# Admin methods
def get_all_users(self):
    """Получить всех пользователей"""
    with self.conn.cursor() as cur:
        cur.execute("SELECT id, username, email, is_admin FROM users ORDER BY id")
        return cur.fetchall()

def update_user(self, user_id, username, email, is_admin):
    """Обновить пользователя"""
    with self.conn.cursor() as cur:
        cur.execute(
            "UPDATE users SET username = %s, email = %s, is_admin = %s WHERE id = %s",
            (username, email, is_admin, user_id)
        )

def delete_user(self, user_id):
    """Удалить пользователя"""
    with self.conn.cursor() as cur:
        cur.execute("DELETE FROM users WHERE id = %s", (user_id,))
```

```
def get_all_hubs(self):
    """Получить все хабы с владельцами"""
    with self.conn.cursor() as cur:
        cur.execute("SELECT h.id, h.user_id, u.username, h.name, h.location, h.serial_n
        return cur.fetchall()

def update_hub(self, hub_id, user_id, name, location, serial_number):
    """Обновить хаб"""
    with self.conn.cursor() as cur:
        cur.execute(
            "UPDATE hubs SET user_id = %s, name = %s, location = %s, serial_number = %s
            (user_id, name, location, serial_number, hub_id)
        )

def delete_hub(self, hub_id):
    """Удалить хаб"""
    with self.conn.cursor() as cur:
        cur.execute("DELETE FROM hubs WHERE id = %s", (hub_id,))

def get_all_devices(self):
    """Получить все устройства"""
    with self.conn.cursor() as cur:
        cur.execute("SELECT d.id, d.hub_id, COALESCE(h.name, 'Хаб удален'), d.name, CO
        return cur.fetchall()

def update_device(self, device_id, hub_id, type_id, name, status):
    """Обновить устройство"""
    with self.conn.cursor() as cur:
        cur.execute(
            "UPDATE devices SET hub_id = %s, type_id = %s, name = %s, status = %s WHERE
            (hub_id, type_id, name, status, device_id)
        )

def delete_device_admin(self, device_id):
    """Удалить устройство (админ)"""
    with self.conn.cursor() as cur:
```

```
            cur.execute("DELETE FROM devices WHERE id = %s", (device_id,))

    # Search methods for admin panel
    def get_users_filtered(self, filter_text=""):
        """Получить пользователей с фильтрацией"""
        query = """
            SELECT id, username, email, is_admin FROM users
            WHERE username ILIKE %s OR email ILIKE %s
            ORDER BY id
        """
        with self.conn.cursor() as cur:
            cur.execute(query, (f"%{filter_text}%", f"%{filter_text}%"))
            return cur.fetchall()

    def get_hubs_filtered(self, filter_text=""):
        """Получить хабы с фильтрацией"""
        query = """
            SELECT h.id, h.user_id, u.username, h.name, h.location, h.serial_number
            FROM hubs h
            JOIN users u ON h.user_id = u.id
            WHERE h.name ILIKE %s OR h.location ILIKE %s OR h.serial_number ILIKE %s OR u.u
            ORDER BY h.id
        """
        with self.conn.cursor() as cur:
            cur.execute(query, (f"%{filter_text}%", f"%{filter_text}%", f"%{filter_text}%"
            return cur.fetchall()

    def get_devices_filtered(self, filter_text=""):
        """Получить устройства с фильтрацией"""
        query = """
            SELECT d.id, d.hub_id, COALESCE(h.name, 'Хаб удален'), d.name, COALESCE(dt.type
            FROM devices d
            LEFT JOIN hubs h ON d.hub_id = h.id
            LEFT JOIN device_types dt ON d.type_id = dt.id
            WHERE d.name ILIKE %s OR COALESCE(dt.type_name, '') ILIKE %s OR COALESCE(h.name
            ORDER BY d.id
        """
```

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

Приложение А3. Исходный код dialogs.py

```
import json
from PyQt6.QtWidgets import (
    QDialog,
    QVBoxLayout,
    QFormLayout,
    QDialogButtonBox,
    QMessageBox,
    QLineEdit,
    QComboBox,
    QCheckBox,
    QPushButton,
    QLabel,
)
from database import Database


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 данные пользователя
            with self.db.conn.cursor() as cur:
```

```
                cur.execute(
                    "SELECT id, username, email, is_admin, password_hash FROM users WHERE u
                    (username,),
                )
                user_data = cur.fetchone()

            if user_data and (
                Database.verify_password(password, user_data[4]) or user_data[4] == passwor
            ):
                self.user_id = user_data[0]
                self.is_admin = user_data[3]
                self.accept()
            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 HubDialog(QDialog):
    """Диалог для добавления хаба"""

    def __init__(self, parent=None, db=None, user_id=None):
        super().__init__(parent)
        self.db = db
        self.user_id = user_id
        self.setWindowTitle("Добавить хаб")
        self.setModal(True)
        self.resize(400, 200)

        layout = QVBoxLayout(self)
```

```
        # Создаем форму
        form_layout = QFormLayout()

        self.name_edit = QLineEdit()
        self.name_edit.setPlaceholderText("Введите название хаба")
        self.location_edit = QLineEdit()
        self.location_edit.setPlaceholderText("Введите местоположение")
        self.serial_edit = QLineEdit()
        self.serial_edit.setPlaceholderText("Введите серийный номер")

        form_layout.addRow("Название:", self.name_edit)
        form_layout.addRow("Местоположение:", self.location_edit)
        form_layout.addRow("Серийный номер:", self.serial_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)

    def get_data(self):
        """Возвращает данные из формы"""
        return {
            "name": self.name_edit.text().strip(),
            "location": self.location_edit.text().strip(),
            "serial_number": self.serial_edit.text().strip(),
        }


class UserDialog(QDialog):
    """Диалог для добавления/редактирования пользователя"""

    def __init__(self, parent=None, db=None, user_data=None):
```

```
        super().__init__(parent)
        self.db = db
        self.user_data = user_data  # For editing
        self.setWindowTitle("Добавить пользователя" if user_data is None else "Редактирова
        self.setModal(True)
        self.resize(350, 200)

        layout = QVBoxLayout(self)
        form = QFormLayout()

        self.username_edit = QLineEdit()
        self.email_edit = QLineEdit()
        self.password_edit = QLineEdit()
        self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)
        self.admin_check = QCheckBox("Администратор")

        if user_data:
            self.username_edit.setText(user_data[1])  # username
            self.email_edit.setText(user_data[2])     # email
            self.admin_check.setChecked(user_data[3]) # is_admin
            self.password_edit.setPlaceholderText("Оставьте пустым, чтобы не менять")
        else:
            self.password_edit.setPlaceholderText("Введите пароль")

        form.addRow("Логин:", self.username_edit)
        form.addRow("Email:", self.email_edit)
        form.addRow("Пароль:", self.password_edit)
        form.addRow(self.admin_check)

        layout.addLayout(form)

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

    def get_data(self):
```

```
        """Возвращает данные из формы"""
        return {
            "username": self.username_edit.text().strip(),
            "email": self.email_edit.text().strip(),
            "password": self.password_edit.text(),
            "is_admin": self.admin_check.isChecked(),
        }


class HubAdminDialog(QDialog):
    """Диалог для добавления/редактирования хаба (админ)"""

    def __init__(self, parent=None, db=None, hub_data=None):
        super().__init__(parent)
        self.db = db
        self.hub_data = hub_data  # For editing
        self.setWindowTitle("Добавить хаб" if hub_data is None else "Редактировать хаб")
        self.setModal(True)
        self.resize(400, 200)

        layout = QVBoxLayout(self)
        form = QFormLayout()

        self.user_combo = QComboBox()
        self.name_edit = QLineEdit()
        self.location_edit = QLineEdit()
        self.serial_edit = QLineEdit()

        # Populate users
        users = self.db.get_all_users()
        for user_id, username, email, is_admin in users:
            self.user_combo.addItem(username, user_id)

        if hub_data:
            self.user_combo.setCurrentIndex(self.user_combo.findData(hub_data[1]))  # user_
            self.name_edit.setText(hub_data[3])      # name
            self.location_edit.setText(hub_data[4])  # location
```

```
            # Показываем серийный номер как статический текст
            self.serial_label = QLabel(hub_data[5])  # serial

        form.addRow("Владелец:", self.user_combo)
        form.addRow("Название:", self.name_edit)
        form.addRow("Местоположение:", self.location_edit)
        if hub_data:
            form.addRow("Серийный номер:", self.serial_label)
        else:
            form.addRow("Серийный номер:", self.serial_edit)

        layout.addLayout(form)

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

    def get_data(self):
        """Возвращает данные из формы"""
        return {
            "user_id": self.user_combo.currentData(),
            "name": self.name_edit.text().strip(),
            "location": self.location_edit.text().strip(),
            "serial_number": self.serial_edit.text().strip(),
        }


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

    def __init__(self, parent=None, db=None, device_data=None):
        super().__init__(parent)
        self.db = db
        self.device_data = device_data  # For editing
        self.setWindowTitle("Добавить устройство" if device_data is None else "Редактироват
        self.setModal(True)
```

```
        self.resize(400, 300)

        layout = QVBoxLayout(self)
        form = QFormLayout()

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

        # Populate combos
        hubs = self.db.get_all_hubs()
        for hub_id, user_id, username, name, location, serial in hubs:
            self.hub_combo.addItem(name, hub_id)

        types = self.db.get_device_types()
        for type_id, type_name in types:
            self.type_combo.addItem(type_name, type_id)

        if device_data:
            self.hub_combo.setCurrentIndex(self.hub_combo.findData(device_data[1]))  # hub_
            self.name_edit.setText(device_data[3])    # name
            self.type_combo.setCurrentIndex(self.type_combo.findData(device_data[4]))  # ty
            self.status_edit.setText(device_data[5])  # status

        form.addRow("Хаб:", self.hub_combo)
        form.addRow("Тип:", self.type_combo)
        form.addRow("Название:", self.name_edit)
        form.addRow("Статус (JSON):", self.status_edit)

        layout.addLayout(form)

        buttons = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.St
        buttons.accepted.connect(self.accept)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)
```

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

Приложение А4. Исходный код widgets.py

```
from PyQt6.QtWidgets import QLineEdit
from PyQt6.QtCore import Qt


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)
```