ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | Белая Т.И |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ДОКУМЕНТАЦИИ  Документация для разработчика |
|  |
| по курсу: Проектирование программных систем |
|  |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4133К |  |  |  | Ковалев Д.В |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**Оглавление**

[**Назначение и условия применения программы** 3](#_Toc166498492)

[**Характеристики программы** 4](#_Toc166498493)

[**Вызов программы** 5](#_Toc166498494)

[**Входные и выходные данные** 5](#_Toc166498495)

[**Сообщения** 6](#_Toc166498496)

[**Логика работы программы, функции и методы** 7](#_Toc166498497)

[**Приложения** 15](#_Toc166498498)

# **Назначение и условия применения программы**

**Назначение программы**:

Программа предназначена для управления процессами при охране территорий с использованием дронов и датчиков. Основная цель - обеспечение эффективных охранных мероприятий, взаимодействия операторов дронов, охраны и административным персоналом через автоматизированную систему управления секторами, датчиками, дронами, событиями, а также получение и обработка информации о секторах.

**Основные функции программы:**

1. **Управление секторами, датчиками, дронами, событиями ипользователями:** 
   * Создание новых данных о секторах, датчиках, дронах, событиях и пользователях.
   * Просмотр списков всех секторов, датчиков, дронов, событий и пользователей.
   * Изменение действительной информации, конкретных о секторах, датчиках, дронах, событиях и пользователях.
   * Удаление информации о секторах, датчиках, дронах, событиях и пользователях.
2. **Работа с центром информации осекторах:** 
   * Получение расширенной информации о выбранном секторе
   * Просмотр всех элементов, принадлежащих сектору и подробной информации о них
   * Получение текстовых описаний о секторе
   * Генерация отчетов по сектору со всеми вытекающими элементами
3. **Работа операторов дронов:**
   * Получение информации по выбранному сектору и дрону
   * Получение видеопотока с работающего дрона
   * Получение информации о сеансе подключения к дрону
   * Непосредственная работа с дроном
4. **Возможности безопасности:** 
   * Настройка параметров системы, включая управление доступом пользователей к различным функциям.
   * Возможность добавления и настройки ролей пользователей для разграничения доступа к информации и операциям в системе.

**Условия применения:**

• Программа устанавливается на ЭВМ машину, поддерживающую C++ , PosgreSQL, QT Creator что обеспечивает ее масштабируемость и высокую производительность.

• Для использования программы необходимо наличие вычислительных мощностей

• Доступ к программе контролируется системой авторизации и аутентификации, которая гарантирует защиту и конфиденциальность данных пользователей.

• Программа предоставляет различные роли для пользователей, что обеспечивает дополнительную безопасность и удобство при работе с системой.

# **Характеристики программы**

Каждое рабочее место должно быть оборудовано компьютером с следующими техническими характеристиками: процессор Intel Core i7-9700, 12 ГБ оперативной памяти, 1 ТБ жесткого диска и т.д. Компьютер должен быть подключен к локальной сети и интернету через Wi-Fi роутер Asus RT-AC86U. В качестве операционной системы используется Windows 10. На компьютерах должна быть установлена среда выполнения С++ и среда работы с базами данных PosgreSQL.

Временные характеристики: Обработка данных происходит в режиме реального времени, реакция на действия пользователя является почти мгновенной. Временной ресурс: Отклик на действия 1-2 секунд.

Режим работы: В зависимости от варианта установки, может быть доступна круглосуточно или в рабочие часы предприятия.

# **Вызов программы**

**Процедуры вызова:** Доступ к программному обеспечению осуществляется через интерфейс. Пользователи могут выполнять такие операции, как CRUD-операции для всех сущностей или доступ к информационному центру полей и странице оператора дрона, через формы, которые внутренне вызывают соответствующие методы обслуживания.

**Методы передачи параметров управления и данных:** данные передаются через connect-запросы, обрабатывая методы для получения и передачи данных.

# **Входные и выходные данные**

**Организация используемой входной информации:**

Входная информация может быть представлена в виде:

* Ввода с клавиатуры
* Использованием оптического манипулятора вида «Мышь»
* Получение видеоданных с дронов и предназначенных устройств

**Выходные данные:**

Выходная информация может быть реализована в виде:

* Текстовых данных
* Обработанных видеопотоков дронов
* Передачи данных в информационный центр системы

**Кодировка:** для текстовых данных применяется кодировка UTF-8, которая обеспечивает совместимость и предотвращает проблемы с кодировкой.

# **Сообщения**

**Обработка ошибок на форме:**

* Поля формы проверяются на соответствие требованиям (например, числовые данные на соответствие действительности, создаваемые сущности на предмет дубликатов, обязательные поля).

**Пользовательский интерфейс для ошибок:**

* Ошибки на форме отображаются поверх полей ввода.
* В случае ошибок валидации поверх полей ввода выводится сообщение об ошибке, в которой говорится как ее исправить.

**Примеры сообщений об ошибках на форме login средствами QMessageBox:**

"Ошибка авторизации", "Неверное имя пользователя или пароль."

Данная ошибка следует из нарушения логики заполнения полей, подробнее можно увидеть на ui форме.

**Обработка ошибок в C++**

**Обработка ошибок в классах**

**Логика обработки:**

• Классы отвечают за реализацию логики и взаимодействуют с базой данных или другими классами. При возникновении ошибок, таких как нарушение правил или проблемы доступа к ресурсам, генерируются исключения.

• Возникшие исключения регистрируются в консоли разработчика.

**Механизмы восстановления после ошибок:**

* В случае обнаружения ошибки классы могут пытаться выполнить альтернативные действия, возвращать стандартные значения.

**Примеры сообщений об ошибках в сервисах с выводом в консоль разработчика:**

* "Ошибка доступа к базе данных. Попробуйте повторить операцию позже."
* "Не удалось найти запрашиваемый ресурс. Возможно, он был удалён."

**Примеры Push уведомлений на ошибки:**

"Ошибка выполнения запроса:"

"Ошибка", "Поле должно содержать только буквы и цифры."

# **Логика работы программы, функции и методы**

Программа состоит из 3 видов файлов:

1. Заголовочные файлы(.h) - Файлы с расширением .h, также известные как заголовочные файлы (header files), используются в языке программирования C и C++ для объявления функций, переменных и констант. Они содержат прототипы функций и объявления структур данных, которые будут использоваться в программе. Обычно .h файлы содержат только объявления (declaration) без определений (definition) функций или переменных.
2. Файлы с исходным кодом (.cpp) - В файлах .cpp содержится реализация функций, классов, переменных и других конструкций, определенных в программе. То есть, в отличие от заголовочных файлов (.h), которые содержат только объявления, .cpp файлы содержат их реализации.
3. Файлы с расширением .ui (.ui) - это файлы пользовательского интерфейса, которые используются в различных инструментах для разработки графических пользовательских интерфейсов (GUI). Одним из самых распространенных инструментов для создания .ui файлов является Qt Designer, который является частью фреймворка Qt для разработки программного обеспечения.

**Заголовочные файлы(.h)**

Все заголовочные файлы похожи между собой, для примера рассмотрим файл login.h отвечающий за объявления класса, библиотек и методов участвующих в процессе авторизации.

Файл login.h - это заголовочный файл (header file), который содержит объявление класса Login (Листинг файла представлен в [приложении 1](#_Приложения)). Класс Login наследует функциональность классов QMainWindow и Ui::Login и дополняет их собственными методами и слотами.

Директива #ifndef LOGIN\_H начинает стражу включения (include guard), которая предотвращает множественное включение файла. Если файл login.h уже был включен в программу, то его содержимое не будет включено повторно.

Директива #define LOGIN\_H определяет метку LOGIN\_H, которая используется в страже включения.

Затем следует включение необходимых заголовочных файлов, в том числе QMainWindow и ui\_login.h.

Далее идет объявление класса Login, который наследует от QMainWindow и Ui::Login. Он содержит открытые, закрытые и защищенные члены класса, включая конструкторы, деструктор, слоты и приватные методы.

Секция private slots: содержит объявление слотов, которые реагируют на сигналы (обычно от пользовательского интерфейса).

Секция private: содержит приватные члены класса, включая указатель ui, который представляет объект пользовательского интерфейса, и приватные методы, такие как openUserWindow(), openAdminWindow() и openAnaliystWindow().

Стража включения #endif // LOGIN\_H завершает страж включения.

**Файлы с исходным кодом (.cpp)**

Все файлы с исходным кодом похожи между собой, для примера рассмотрим файл login.cpp отвечающий за реализацию методов класса авторизации.

Файл login.cpp содержит реализацию методов класса Login (Листинг файла представлен в [приложении 2](#_Приложения)). Этот класс представляет окно входа в систему. Он отвечает за обработку событий, связанных с входом пользователя, а также за открытие различных окон приложения в зависимости от роли пользователя.

Метод handleLoginButtonClicked() вызывается при нажатии на кнопку входа. Он получает введенное пользователем имя и пароль, а затем проверяет их с помощью метода authenticateUser() класса DatabaseManager. В зависимости от роли пользователя окно входа открывает соответствующее окно приложения.

Методы openUserWindow(), openAnaliystWindow() и openAdminWindow() отвечают за открытие окон для пользователя, аналитика и администратора соответственно.

**Файлы с расширением .ui**

Все файлы с расширением .ui похожи между собой структурой, для примера рассмотрим файл login.ui отвечающий за создание пользовательского интерфейса класса авторизации.

Файл login.ui представляет собой XML-файл, который описывает пользовательский интерфейс для окна входа в систему (Листинг файла представлен в [приложении 3](#_Приложения)). Он создан с помощью инструмента Qt Designer или другого редактора форм.

Файл содержит различные элементы интерфейса, такие как окно приложения, метки, поля ввода и кнопки. Эти элементы размещены с помощью различных макетов (layout), таких как QVBoxLayout.

Каждый элемент имеет свои свойства, такие как текст, геометрия, выравнивание и т.д. Например, метки "Логин" и "Пароль" выровнены по левому краю, а поля ввода - по центру.

Сигналы и слоты также определены в файле, чтобы обеспечить связь между элементами интерфейса и функциональностью приложения. Например, сигнал clicked() кнопки "Войти" соединен со слотом on\_loginButton\_clicked(), который обрабатывает событие нажатия кнопки.

(Листинг файла представлен в приложении 4).

Особые классы:

“DatabaseManager”: Этот класс отвечает за взаимодействие с базой данных. Он реализует методы для открытия и закрытия соединения с базой данных, аутентификации пользователей, создания таблиц, добавления данных и извлечения информации из базы данных.

Конфигурация безопасности:

Код класса “DatabaseManager” (Листинг файла представлен в приложении 4).представляет собой конфигурацию безопасности для взаимодействия с базой данных в приложении. Он обеспечивает безопасное хранение паролей пользователей, так как использует хэширование паролей перед их сохранением и сравнением при аутентификации.

Код состоит из нескольких основных частей:

1. Открытие и закрытие базы данных: Класс обеспечивает безопасное открытие и закрытие соединения с базой данных.

2. Аутентификация пользователей: Метод `authenticateUser` выполняет аутентификацию пользователей, проверяя соответствие имени пользователя и хэшированного пароля в базе данных.

3. Создание таблиц: Метод `createTables` создает несколько таблиц в базе данных для хранения информации о пользователях, дронах, секторах патрулирования, сенсорах и событиях.

4. Добавление данных: Класс предоставляет методы для добавления пользователей, дронов, секторов патрулирования, сенсоров и событий в базу данных.

5. Извлечение данных: Класс также реализует методы для извлечения информации о секторах патрулирования, дронах, сенсорах и событиях из базы данных.Этот файл необходим для обеспечения безопасности приложения, а так же обеспечения работы с базой данных. Он позволяет контролировать доступ к различным частям приложения в зависимости от роли пользователя.

# **Приложения**

**Приложение 1**

#ifndef LOGIN\_H

#define LOGIN\_H

#include <QMainWindow>

#include "ui\_login.h"

class Login : public QMainWindow, public Ui::Login

{

Q\_OBJECT

public:

explicit Login(QWidget \*parent = nullptr);

~Login();

private slots:

void handleLoginButtonClicked();

private:

Ui::Login \*ui;

void openUserWindow();

void openAdminWindow();

void openAnaliystWindow();

};

#endif // LOGIN\_H

**Приложение 2**

#include "login.h"

#include "ui\_login.h"

#include "databasemanager.h"

#include "operatorwindow.h"

#include <QMessageBox>

#include "adminwindow.h"

#include "userwindow.h"

#include "analystwindow.h"

Login::Login(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Login)

{

ui->setupUi(this);

// Подключение сигнала кнопки к слоту

connect(ui->loginButton, &QPushButton::clicked, this, &Login::handleLoginButtonClicked);

}

Login::~Login()

{

delete ui;

}

void Login::handleLoginButtonClicked()

{

QString username = ui->usernameEdit->text();

QString password = ui->passwordEdit->text();

QString role = DatabaseManager::instance().authenticateUser(username, password);

if (!role.isEmpty()) {

if (role == "Администратор") {

openAdminWindow();

} else if (role == "Оператор") {

openUserWindow();

} else if (role == "Аналитик") {

openAnaliystWindow();

}

this->close();

} else {

QMessageBox::warning(this, "Ошибка авторизации", "Неверное имя пользователя или пароль.");

}

}

void Login::openUserWindow()

{

UserWindow \*userWindow = new UserWindow();

userWindow->show();

// Открытие окна оператора

OperatorWindow \*operatorWindow = new OperatorWindow();

operatorWindow->show();

}

void Login::openAnaliystWindow()

{

AnalystWindow \*analystWindow = new AnalystWindow();

analystWindow->show();

}

void Login::openAdminWindow()

{

AdminWindow \*adminWindow = new AdminWindow();

adminWindow->show();

// Открытие окна оператора

// OperatorWindow \*operatorWindow = new OperatorWindow();

//operatorWindow->show();

}

**Приложение 3**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>Login</class>

<widget class="QMainWindow" name="Login">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>545</width>

<height>332</height>

</rect>

</property>

<property name="windowTitle">

<string>Login</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<spacer name="verticalSpacer">

<property name="orientation">

<enum>Qt::Vertical</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>0</width>

<height>0</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QLabel" name="titleLabel">

<property name="text">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:14pt; color:black; font-weight:bold;&quot;&gt;Система управления ISOD&lt;/span&gt;&lt;/p&gt;&lt;p align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:10pt; color:#ff0000;&quot;&gt;Авторизуйтесь с помощью ваших персональных данных&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item alignment="Qt::AlignLeft">

<widget class="QLabel" name="usernameLabel">

<property name="text">

<string>Логин:</string>

</property>

<property name="alignment">

<set>Qt::AlignRight</set>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="usernameEdit">

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item alignment="Qt::AlignLeft">

<widget class="QLabel" name="passwordLabel">

<property name="text">

<string>Пароль:</string>

</property>

<property name="alignment">

<set>Qt::AlignRight</set>

</property>

</widget>

</item>

<item>

<widget class="QLineEdit" name="passwordEdit">

<property name="echoMode">

<enum>QLineEdit::Password</enum>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="loginButton">

<property name="text">

<string>Войти</string>

</property>

</widget>

</item>

<item>

<spacer name="verticalSpacer\_2">

<property name="orientation">

<enum>Qt::Vertical</enum>

</property>

<property name="sizeHint" stdset="0">

<size>

<width>0</width>

<height>0</height>

</size>

</property>

</spacer>

</item>

<item>

<widget class="QLabel" name="contactLabel">

<property name="text">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span style=&quot; font-size:8pt;&quot;&gt;При утере данных или технических проблемах обратитесь по номеру: 8-953-344-14-93&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>

</property>

<property name="alignment">

<set>Qt::AlignBottom|Qt::AlignRight|Qt::AlignTrailing</set>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>545</width>

<height>26</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections>

<connection>

<sender>loginButton</sender>

<signal>clicked()</signal>

<receiver>Login</receiver>

<slot>on\_loginButton\_clicked()</slot>

<hints>

<hint type="sourcelabel">

<x>420</x>

<y>237</y>

</hint>

<hint type="destinationlabel">

<x>295</x>

<y>197</y>

</hint>

</hints>

</connection>

</connections>

<slots>

<slot>on\_loginButton\_clicked()</slot>

</slots>

</ui>

**Приложение 4**

#include "databasemanager.h"

#include <QCryptographicHash>

DatabaseManager::DatabaseManager() {

// Инициализация базы данных и других членов класса

}

DatabaseManager::~DatabaseManager() {

closeDatabase();

QSqlDatabase::removeDatabase(QSqlDatabase::defaultConnection);

}

DatabaseManager& DatabaseManager::instance() {

static DatabaseManager instance;

return instance;

}

bool DatabaseManager::openDatabase() {

if (!QSqlDatabase::contains(QSqlDatabase::defaultConnection)) {

m\_database = QSqlDatabase::addDatabase("QPSQL");

} else {

m\_database = QSqlDatabase::database(QSqlDatabase::defaultConnection);

}

m\_database.setHostName("localhost");

m\_database.setPort(5432);

m\_database.setDatabaseName("ISAD");

m\_database.setUserName("postgres");

m\_database.setPassword("root");

return m\_database.open();

}

void DatabaseManager::closeDatabase() {

if (m\_database.isOpen()) {

m\_database.close();

}

}

QString DatabaseManager::authenticateUser(const QString &username, const QString &password) {

if (!m\_database.isOpen() && !openDatabase()) {

return QString();

}

QSqlQuery query;

query.prepare("SELECT Role FROM Users WHERE Username = :username AND Password = :password");

query.bindValue(":username", username);

// Хэширование пароля перед выполнением запроса

QByteArray hashedPassword = QCryptographicHash::hash(password.toUtf8(), QCryptographicHash::Md5);

query.bindValue(":password", QString(hashedPassword.toHex()));

if (query.exec() && query.next()) {

return query.value(0).toString();

} else {

return QString();

}

}

bool DatabaseManager::createTables() {

if (!m\_database.isOpen() && !openDatabase()) {

return false;

}

QSqlQuery query(m\_database);

bool success = true;

// Создание таблицы пользователей

success = query.exec("CREATE TABLE IF NOT EXISTS Users ("

"ID SERIAL PRIMARY KEY,"

"Username VARCHAR(255) NOT NULL UNIQUE,"

"Password VARCHAR(255) NOT NULL,"

"Role VARCHAR(50)"

");");

if (!success) {

return false;

}

// Добавление главного администратора

success = addUser("KovalevD", "18062003", "Администратор");

if (!success) {

return false;

}

// Создание таблицы беспилотных устройств

success = query.exec("CREATE TABLE IF NOT EXISTS Drones ("

"ID SERIAL PRIMARY KEY,"

"Name VARCHAR(255) NOT NULL,"

"Type VARCHAR(255) NOT NULL,"

"UserID INTEGER REFERENCES Users(ID)"

");");

if (!success) {

return false;

}

// Создание таблицы секторов патрулирования

success = query.exec("CREATE TABLE IF NOT EXISTS PatrolSectors ("

"ID SERIAL PRIMARY KEY,"

"Name VARCHAR(255) NOT NULL,"

"Description TEXT"

");");

if (!success) {

return false;

}

// Создание таблицы связи между дронами и секторами патрулирования

success = query.exec("CREATE TABLE IF NOT EXISTS DronePatrols ("

"DroneID INTEGER REFERENCES Drones(ID),"

"SectorID INTEGER REFERENCES PatrolSectors(ID),"

"PRIMARY KEY (DroneID, SectorID),"

"TakeoffTimestamp TIMESTAMP,"

"TimeEvent TIMESTAMP"

");");

if (!success) {

return false;

}

// Создание таблицы сенсоров

success = query.exec("CREATE TABLE IF NOT EXISTS Sensors ("

"ID SERIAL PRIMARY KEY,"

"Type VARCHAR(255) NOT NULL,"

"Description TEXT"

");");

if (!success) {

return false;

}

// Создание таблицы связи между дронами и сенсорами

success = query.exec("CREATE TABLE IF NOT EXISTS DroneSensors ("

"DroneID INTEGER REFERENCES Drones(ID),"

"SensorID INTEGER REFERENCES Sensors(ID),"

"PRIMARY KEY (DroneID, SensorID)"

");");

if (!success) {

return false;

}

// Создание таблицы событий

success = query.exec("CREATE TABLE IF NOT EXISTS Events ("

"ID SERIAL PRIMARY KEY,"

"EventType VARCHAR(255) NOT NULL,"

"EventDescription TEXT"

");");

if (!success) {

return false;

}

// Создание таблицы связи между событиями и дронами

success = query.exec("CREATE TABLE IF NOT EXISTS DroneEvent ("

"EventID INTEGER REFERENCES Events(ID),"

"DroneID INTEGER REFERENCES Drones(ID),"

"PRIMARY KEY (EventID, DroneID),"

"TimeEvent TIMESTAMP"

");");

if (!success) {

return false;

}

return true;

}

bool DatabaseManager::addUser(const QString &username, const QString &password, const QString &role) {

if (!m\_database.isOpen() && !openDatabase()) {

return false;

}

QSqlQuery query(m\_database);

query.prepare("INSERT INTO Users (Username, Password, Role) VALUES (:username, :password, :role)");

// Хэширование пароля перед добавлением в базу данных

QByteArray hashedPassword = QCryptographicHash::hash(password.toUtf8(), QCryptographicHash::Md5);

query.bindValue(":username", username);

query.bindValue(":password", QString(hashedPassword.toHex()));

query.bindValue(":role", role);

return query.exec();

}

bool DatabaseManager::addDrone(const QString &name, const QString &type, const QString &userid) {

if (!m\_database.isOpen() && !openDatabase()) {

return false;

}

QSqlQuery query(m\_database);

query.prepare("INSERT INTO Drones (name, type, userid) VALUES (:name, :type, :userid)");

query.bindValue(":name", name);

query.bindValue(":type", type);

query.bindValue(":userid", userid);

return query.exec();

}

bool DatabaseManager::addSectors(const QString &name, const QString &description) {

if (!m\_database.isOpen() && !openDatabase()) {

return false;

}

QSqlQuery query(m\_database);

query.prepare("INSERT INTO patrolsectors (name, description ) VALUES (:name, :description)");

query.bindValue(":name", name);

query.bindValue(":description", description);

return query.exec();

}

bool DatabaseManager::addSensors(const QString &type, const QString &description) {

if (!m\_database.isOpen() && !openDatabase()) {

return false;

}

QSqlQuery query(m\_database);

query.prepare("INSERT INTO Sensors (type, description ) VALUES (:type, :description)");

query.bindValue(":type", type);

query.bindValue(":description", description);

return query.exec();

}

QVector<QVector<QString>> DatabaseManager::getPatrolSectors()

{

QVector<QVector<QString>> sectorsData;

if (!m\_database.isOpen() && !openDatabase()) {

return sectorsData;

}

QSqlQuery query;

if (!query.exec("SELECT \* FROM PatrolSectors")) {

return sectorsData;

}

while (query.next()) {

QVector<QString> sector;

sector.append(query.value(0).toString()); // ID

sector.append(query.value(1).toString()); // Name

sector.append(query.value(2).toString()); // Description

sectorsData.append(sector);

}

return sectorsData;

}

QVector<QVector<QString>> DatabaseManager::getDroneSensors()

{

QVector<QVector<QString>> sensorsData;

if (!m\_database.isOpen() && !openDatabase()) {

return sensorsData;

}

QSqlQuery query;

if (!query.exec("SELECT \* FROM Sensors")) {

return sensorsData;

}

while (query.next()) {

QVector<QString> sensor;

sensor.append(query.value(0).toString()); // ID

sensor.append(query.value(1).toString()); // Type

sensor.append(query.value(2).toString()); // Description

sensorsData.append(sensor);

}

return sensorsData;

}

QVector<QVector<QString>> DatabaseManager::getDronePatrols()

{

QVector<QVector<QString>> patrolsData;

if (!m\_database.isOpen() && !openDatabase()) {

return patrolsData;

}

QSqlQuery query;

if (!query.exec("SELECT \* FROM DronePatrols")) {

return patrolsData;

}

while (query.next()) {

QVector<QString> patrol;

patrol.append(query.value(0).toString()); // DroneID

patrol.append(query.value(1).toString()); // SectorID

patrol.append(query.value(2).toString()); // TakeoffTimestamp

patrol.append(query.value(3).toString()); // TimeEvent

patrolsData.append(patrol);

}

return patrolsData;

}

QVector<QVector<QString>> DatabaseManager::getDroneEvents()

{

QVector<QVector<QString>> eventsData;

if (!m\_database.isOpen() && !openDatabase()) {

return eventsData;

}

QSqlQuery query;

if (!query.exec("SELECT \* FROM DroneEvent")) {

return eventsData;

}

while (query.next()) {

QVector<QString> event;

event.append(query.value(0).toString()); // EventID

event.append(query.value(1).toString()); // DroneID

event.append(query.value(2).toString()); // TimeEvent

eventsData.append(event);

}

return eventsData;

}

QVector<QVector<QString>> DatabaseManager::getDroneData() {

QVector<QVector<QString>> droneData;

if (!m\_database.isOpen() && !openDatabase()) {

return droneData;

}

QSqlQuery query;

if (!query.exec("SELECT ID, Name, Type,Userid FROM Drones")) {

qDebug() << "Failed to fetch drone data:" << query.lastError().text();

return droneData;

}

while (query.next()) {

QVector<QString> droneRow;

droneRow.push\_back(query.value(0).toString()); // ID

droneRow.push\_back(query.value(1).toString()); // Name

droneRow.push\_back(query.value(2).toString());// Type

droneRow.push\_back(query.value(3).toString());//Userid

droneData.push\_back(droneRow);

}

qDebug() << "Drone data fetched successfully:" << droneData;

return droneData;

}

QVector<QVector<QString>> DatabaseManager::getUserData() {

QVector<QVector<QString>> userData;

if (!m\_database.isOpen() && !openDatabase()) {

return userData;

}

QSqlQuery query;

if (!query.exec("SELECT ID, Username, Role FROM Users")) {

qDebug() << "Failed to fetch user data:" << query.lastError().text();

return userData;

}

while (query.next()) {

QVector<QString> userRow;

userRow.push\_back(query.value(0).toString()); // ID

userRow.push\_back(query.value(1).toString()); // Username

userRow.push\_back(query.value(2).toString()); // Role

userData.push\_back(userRow);

}

return userData;

}

QVector<QVector<QString>> DatabaseManager::getSectorData() {

QVector<QVector<QString>> sectorData;

if (!m\_database.isOpen() && !openDatabase()) {

return sectorData;

}

QSqlQuery query;

if (!query.exec("SELECT ID, Name, Description FROM PatrolSectors")) {

qDebug() << "Failed to fetch sector data:" << query.lastError().text();

return sectorData;

}

while (query.next()) {

QVector<QString> sectorRow;

sectorRow.push\_back(query.value(0).toString()); // ID

sectorRow.push\_back(query.value(1).toString()); // Name

sectorRow.push\_back(query.value(2).toString()); // Description

sectorData.push\_back(sectorRow);

}

return sectorData;

}

QVector<QVector<QString>> DatabaseManager::getDeviceData() {

QVector<QVector<QString>> deviceData;

if (!m\_database.isOpen() && !openDatabase()) {

return deviceData;

}

QSqlQuery query;

if (!query.exec("SELECT ID, Type, Description FROM Sensors")) {

qDebug() << "Failed to fetch device data:" << query.lastError().text();

return deviceData;

}

while (query.next()) {

QVector<QString> deviceRow;

deviceRow.push\_back(query.value(0).toString()); // ID

deviceRow.push\_back(query.value(1).toString()); // Type

deviceRow.push\_back(query.value(2).toString()); // Description

deviceData.push\_back(deviceRow);

}

return deviceData;

}

QVector<QVector<QString>> DatabaseManager::getEventData() {

QVector<QVector<QString>> eventData;

if (!m\_database.isOpen() && !openDatabase()) {

return eventData;

}

QSqlQuery query;

if (!query.exec("SELECT ID, eventtype, eventdescription FROM events")) {

qDebug() << "Failed to fetch device data:" << query.lastError().text();

return eventData;

}

while (query.next()) {

QVector<QString> eventRow;

eventRow.push\_back(query.value(0).toString()); // ID

eventRow.push\_back(query.value(1).toString()); // Type

eventRow.push\_back(query.value(2).toString()); // Description

eventData.push\_back(eventRow);

}

return eventData;

}

**Приложение 5**

#ifndef DATABASEMANAGER\_H

#define DATABASEMANAGER\_H

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlError>

#include <QVector>

class AdminWindow; // Предварительное объявление класса AdminWindow

class DatabaseManager {

public:

static DatabaseManager& instance(); // Метод для получения единственного экземпляра DatabaseManager

bool openDatabase();

void closeDatabase();

QString authenticateUser(const QString &username, const QString &password);

bool createTables();

bool addUser(const QString &username, const QString &password, const QString &role);

bool addDrone(const QString &name, const QString &type, const QString &userid);

bool addSectors(const QString &name, const QString &description);

bool addSensors(const QString &type, const QString &description);

QVector<QVector<QString>> getPatrolSectors();

QVector<QVector<QString>> getDroneSensors();

QVector<QVector<QString>> getDronePatrols();

QVector<QVector<QString>> getDroneEvents();

QVector<QVector<QString>> getDroneData();

QVector<QVector<QString>> getUserData();

QVector<QVector<QString>> getSectorData();

QVector<QVector<QString>> getDeviceData();

QVector<QVector<QString>> getEventData();

private:

DatabaseManager();

~DatabaseManager(); // Деструктор для закрытия и удаления соединения

DatabaseManager(const DatabaseManager&) = delete;

DatabaseManager& operator=(const DatabaseManager&) = delete;

QSqlDatabase m\_database;

// Дружественные классы

friend class Login;

friend class UserWindow;

friend class AdminWindow;

friend class InformationHub;

friend class AnalystWindow;

};

#endif // DATABASEMANAGER\_H

**Приложение 6**

#include "adminwindow.h"

#include "ui\_adminwindow.h"

#include <QInputDialog>

#include <QRegExp>

#include <QSqlQuery>

AdminWindow::AdminWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::AdminWindow)

{

ui->setupUi(this);

this->setWindowTitle("Окно Администрирования Системы");

setupModels();

setupTableViews();

}

AdminWindow::~AdminWindow()

{

delete ui;

}

void AdminWindow::setupModels()

{

usersModel = new QSqlTableModel(this);

usersModel->setTable("Users");

QSqlQuery query;

query.prepare("SELECT \* FROM Users");

if (query.exec()) {

usersModel->setQuery("SELECT \* FROM Users");

ui->usersTableView->setModel(usersModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

dronesModel = new QSqlTableModel(this);

dronesModel->setTable("Drones");

query.prepare("SELECT \* FROM Drones");

if (query.exec()) {

dronesModel->setQuery("SELECT \* FROM Drones");

ui->dronesTableView->setModel(dronesModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

sectorsModel = new QSqlTableModel(this);

sectorsModel->setTable("PatrolSectors");

query.prepare("SELECT \* FROM PatrolSectors");

if (query.exec()) {

sectorsModel->setQuery("SELECT \* FROM PatrolSectors");

ui->sectorsTableView->setModel(sectorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

sensorsModel = new QSqlTableModel(this);

sensorsModel->setTable("Sensors");

query.prepare("SELECT \* FROM Sensors");

if (query.exec()) {

sensorsModel->setQuery("SELECT \* FROM Sensors");

ui->sensorsTableView->setModel(sensorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

void AdminWindow::setupTableViews()

{

ui->usersTableView->setModel(usersModel);

ui->dronesTableView->setModel(dronesModel);

ui->sectorsTableView->setModel(sectorsModel);

ui->sensorsTableView->setModel(sensorsModel);

}

bool AdminWindow::validateInput(const QString &input)

{

if (input.isEmpty()) {

QMessageBox::warning(this, "Ошибка", "Поле не должно быть пустым.");

return false;

}

QRegExp re("[A-Za-zА-Яёа-я0-9]+");

if (!re.exactMatch(input)) {

QMessageBox::warning(this, "Ошибка", "Поле должно содержать только буквы и цифры.");

return false;

}

return true;

}

void AdminWindow::on\_addButton\_clicked()

{

QString tableName = ui->tabWidget->currentWidget()->objectName();

if (tableName == "tabUsers") {

// Добавление пользователя

QString username = QInputDialog::getText(this, "Добавить пользователя", "Имя пользователя:");

QString password = QInputDialog::getText(this, "Добавить пользователя", "Пароль:", QLineEdit::Password);

QString role = QInputDialog::getText(this, "Добавить пользователя", "Роль:");

if (validateInput(username) && validateInput(password) && validateInput(role)) {

if (!DatabaseManager::instance().addUser(username, password, role)) {

QMessageBox::warning(this, "Ошибка", "Не удалось добавить пользователя.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM Users");

if (query.exec()) {

usersModel->setQuery("SELECT \* FROM Users");

ui->usersTableView->setModel(usersModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabDrones") {

// Добавление дрона

QString name = QInputDialog::getText(this, "Добавить дрон", "Имя:");

QString type = QInputDialog::getText(this, "Добавить дрон", "Тип:");

QString userid = QInputDialog::getText(this, "Добавить дрон", "Оператор:");

if (validateInput(name) && validateInput(type)&& validateInput(userid)) {

if (!DatabaseManager::instance().addDrone(name, type, userid)) {

QMessageBox::warning(this, "Ошибка", "Не удалось добавить пользователя.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM drones");

if (query.exec()) {

dronesModel->setQuery("SELECT \* FROM drones");

ui->dronesTableView->setModel(dronesModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabSectors") {

// Добавление сектора патрулирования

QString name = QInputDialog::getText(this, "Добавить сектор патрулирования", "Имя:");

QString description = QInputDialog::getText(this, "Добавить сектор патрулирования", "Описание:");

if (validateInput(name) && validateInput(description)) {

if (!DatabaseManager::instance().addSectors(name, description)) {

QMessageBox::warning(this, "Ошибка", "Не удалось добавить пользователя.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM patrolsectors");

if (query.exec()) {

sectorsModel->setQuery("SELECT \* FROM patrolsectors");

ui->sectorsTableView->setModel(sectorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabSensors") {

// Добавление сенсора

QString type = QInputDialog::getText(this, "Добавить сенсор", "Тип:");

QString description = QInputDialog::getText(this, "Добавить сенсор", "Описание:");

if (validateInput(type) && validateInput(description)) {

if (!DatabaseManager::instance().addSensors(type, description)) {

QMessageBox::warning(this, "Ошибка", "Не удалось добавить пользователя.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM sensors");

if (query.exec()) {

sensorsModel->setQuery("SELECT \* FROM sensors");

ui->sensorsTableView->setModel(sensorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

}

}

void AdminWindow::on\_editButton\_clicked()

{

QString tableName = ui->tabWidget->currentWidget()->objectName();

QModelIndex currentIndex;

QSqlTableModel \*currentModel;

if (tableName == "tabUsers") {

currentIndex = ui->usersTableView->currentIndex();

currentModel = usersModel;

} else if (tableName == "tabDrones") {

currentIndex = ui->dronesTableView->currentIndex();

currentModel = dronesModel;

} else if (tableName == "tabSectors") {

currentIndex = ui->sectorsTableView->currentIndex();

currentModel = sectorsModel;

} else if (tableName == "tabSensors") {

currentIndex = ui->sensorsTableView->currentIndex();

currentModel = sensorsModel;

}

if (!currentIndex.isValid()) {

QMessageBox::warning(this, "Ошибка", "Выберите запись для редактирования.");

return;

}

if (tableName == "tabUsers") {

QString username = QInputDialog::getText(this, "Изменить пользователя", "Имя пользователя:");

QString password = QInputDialog::getText(this, "Изменить пользователя", "Пароль:", QLineEdit::Password);

QString role = QInputDialog::getText(this, "Изменить пользователя", "Роль:");

if (validateInput(username) && validateInput(password) && validateInput(role)) {

QSqlRecord record = currentModel->record(currentIndex.row());

record.setValue("Username", username);

record.setValue("Password", QCryptographicHash::hash(password.toUtf8(), QCryptographicHash::Md5).toHex());

record.setValue("Role", role);

if (!currentModel->setRecord(currentIndex.row(), record)) {

//QMessageBox::warning(this, "Ошибка", "Не удалось изменить пользователя.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM Users");

if (query.exec()) {

usersModel->setQuery("SELECT \* FROM Users");

ui->usersTableView->setModel(usersModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabDrones") {

QString name = QInputDialog::getText(this, "Изменить дрон", "Имя:");

QString type = QInputDialog::getText(this, "Изменить дрон", "Тип:");

if (validateInput(name) && validateInput(type)) {

QSqlRecord record = currentModel->record(currentIndex.row());

record.setValue("Name", name);

record.setValue("Type", type);

if (!currentModel->setRecord(currentIndex.row(), record)) {

//QMessageBox::warning(this, "Ошибка", "Не удалось изменить дрон.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM Drones");

if (query.exec()) {

dronesModel->setQuery("SELECT \* FROM Drones");

ui->dronesTableView->setModel(dronesModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabSectors") {

QString name = QInputDialog::getText(this, "Изменить сектор патрулирования", "Имя:");

QString description = QInputDialog::getText(this, "Изменить сектор патрулирования", "Описание:");

if (validateInput(name) && validateInput(description)) {

QSqlRecord record = currentModel->record(currentIndex.row());

record.setValue("Name", name);

record.setValue("Description", description);

if (!currentModel->setRecord(currentIndex.row(), record)) {

//QMessageBox::warning(this, "Ошибка", "Не удалось изменить сектор патрулирования.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM PatrolSectors");

if (query.exec()) {

sectorsModel->setQuery("SELECT \* FROM PatrolSectors");

ui->sectorsTableView->setModel(sectorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

} else if (tableName == "tabSensors") {

QString type = QInputDialog::getText(this, "Изменить сенсор", "Тип:");

QString description = QInputDialog::getText(this, "Изменить сенсор", "Описание:");

if (validateInput(type) && validateInput(description)) {

QSqlRecord record = currentModel->record(currentIndex.row());

record.setValue("Type", type);

record.setValue("Description", description);

if (!currentModel->setRecord(currentIndex.row(), record)) {

//QMessageBox::warning(this, "Ошибка", "Не удалось изменить сенсор.");

} else {

QSqlQuery query;

query.prepare("SELECT \* FROM Sensors");

if (query.exec()) {

sensorsModel->setQuery("SELECT \* FROM Sensors");

ui->sensorsTableView->setModel(sensorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

}

}

}

void AdminWindow::on\_deleteButton\_clicked()

{

QString tableName = ui->tabWidget->currentWidget()->objectName();

QModelIndex currentIndex;

QString idColumn;

QString table;

QSqlTableModel \*currentModel;

if (tableName == "tabUsers") {

currentIndex = ui->usersTableView->currentIndex();

currentModel = usersModel;

idColumn = "id";

table = "Users";

} else if (tableName == "tabDrones") {

currentIndex = ui->dronesTableView->currentIndex();

currentModel = dronesModel;

idColumn = "ID";

table = "Drones";

} else if (tableName == "tabSectors") {

currentIndex = ui->sectorsTableView->currentIndex();

currentModel = sectorsModel;

idColumn = "ID";

table = "PatrolSectors";

} else if (tableName == "tabSensors") {

currentIndex = ui->sensorsTableView->currentIndex();

currentModel = sensorsModel;

idColumn = "ID";

table = "Sensors";

}

if (!currentIndex.isValid()) {

QMessageBox::warning(this, "Ошибка", "Выберите запись для удаления.");

return;

}

if (QMessageBox::question(this, "Удалить запись", "Вы уверены, что хотите удалить эту запись?") == QMessageBox::Yes) {

int id = currentModel->data(currentModel->index(currentIndex.row(), 0)).toInt();

// Проверка значения id

qDebug() << "ID для удаления:" << id;

QSqlQuery query;

QString queryString = QString("DELETE FROM %1 WHERE %2 = %3").arg(table, idColumn, QString::number(id));

// Проверка строки запроса

qDebug() << "Запрос на удаление:" << queryString;

if (!query.exec(queryString)) {

QMessageBox::warning(this, "Ошибка", "Не удалось удалить запись: " + query.lastError().text());

// Проверка ошибки SQL-запроса

qDebug() << "Ошибка SQL-запроса:" << query.lastError().text();

} else {

// Обновление модели после удаления

if (tableName == "tabUsers") {

usersModel->setQuery("SELECT \* FROM Users");

ui->usersTableView->setModel(usersModel);

} else if (tableName == "tabDrones") {

dronesModel->setQuery("SELECT \* FROM Drones");

ui->dronesTableView->setModel(dronesModel);

} else if (tableName == "tabSectors") {

sectorsModel->setQuery("SELECT \* FROM PatrolSectors");

ui->sectorsTableView->setModel(sectorsModel);

} else if (tableName == "tabSensors") {

sensorsModel->setQuery("SELECT \* FROM Sensors");

ui->sensorsTableView->setModel(sensorsModel);

}

QMessageBox::information(this, "Успех", "Запись успешно удалена.");

}

}

}

void AdminWindow::on\_refreshButton\_clicked()

{

QSqlQuery query;

query.prepare("SELECT \* FROM Users");

if (query.exec()) {

usersModel->setQuery("SELECT \* FROM Users");

ui->usersTableView->setModel(usersModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

query.prepare("SELECT \* FROM Drones");

if (query.exec()) {

dronesModel->setQuery("SELECT \* FROM Drones");

ui->dronesTableView->setModel(dronesModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

query.prepare("SELECT \* FROM PatrolSectors");

if (query.exec()) {

sectorsModel->setQuery("SELECT \* FROM PatrolSectors");

ui->sectorsTableView->setModel(sectorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

query.prepare("SELECT \* FROM Sensors");

if (query.exec()) {

sensorsModel->setQuery("SELECT \* FROM Sensors");

ui->sensorsTableView->setModel(sensorsModel);

} else {

qDebug() << "Ошибка выполнения запроса:" << query.lastError().text();

}

}

void AdminWindow::on\_logoutButton\_clicked()

{

this->close();

Login \*loginWindow = new Login();

loginWindow->show();

}

**Приложение 7**

#ifndef ADMINWINDOW\_H

#define ADMINWINDOW\_H

#include "databasemanager.h"

#include "login.h"

#include <QMainWindow>

#include <QSqlTableModel>

#include <QMessageBox>//>

#include <QSqlRecord>

#include <QSqlRelationalDelegate>

#include <QCryptographicHash>

namespace Ui {

class AdminWindow;

}

class AdminWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit AdminWindow(QWidget \*parent = nullptr);

~AdminWindow();

private slots:

void on\_addButton\_clicked();

void on\_editButton\_clicked();

void on\_deleteButton\_clicked();

void on\_refreshButton\_clicked();

void on\_logoutButton\_clicked();

private:

Ui::AdminWindow \*ui;

QSqlTableModel \*usersModel;

QSqlTableModel \*dronesModel;

QSqlTableModel \*sectorsModel;

QSqlTableModel \*sensorsModel;

void setupModels();

void setupTableViews();

bool validateInput(const QString &input);

};

#endif // ADMINWINDOW\_H

**Приложение 8**

<ui version="4.0">

<class>AdminWindow</class>

<widget class="QMainWindow" name="AdminWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>Окно Администрирования Системы</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QTabWidget" name="tabWidget">

<property name="geometry">

<rect>

<x>10</x>

<y>10</y>

<width>780</width>

<height>500</height>

</rect>

</property>

<property name="currentIndex">

<number>0</number>

</property>

<widget class="QWidget" name="tabUsers">

<attribute name="title">

<string>Пользователи</string>

</attribute>

<widget class="QTableView" name="usersTableView">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>780</width>

<height>480</height>

</rect>

</property>

</widget>

</widget>

<widget class="QWidget" name="tabDrones">

<attribute name="title">

<string>Дроны</string>

</attribute>

<widget class="QTableView" name="dronesTableView">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>780</width>

<height>480</height>

</rect>

</property>

</widget>

</widget>

<widget class="QWidget" name="tabSectors">

<attribute name="title">

<string>Сектора Патрулирования</string>

</attribute>

<widget class="QTableView" name="sectorsTableView">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>780</width>

<height>480</height>

</rect>

</property>

</widget>

</widget>

<widget class="QWidget" name="tabSensors">

<attribute name="title">

<string>Сенсоры</string>

</attribute>

<widget class="QTableView" name="sensorsTableView">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>780</width>

<height>480</height>

</rect>

</property>

</widget>

</widget>

</widget>

<widget class="QPushButton" name="addButton">

<property name="geometry">

<rect>

<x>10</x>

<y>520</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Добавить</string>

</property>

</widget>

<widget class="QPushButton" name="editButton">

<property name="geometry">

<rect>

<x>100</x>

<y>520</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Изменить</string>

</property>

</widget>

<widget class="QPushButton" name="deleteButton">

<property name="geometry">

<rect>

<x>190</x>

<y>520</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Удалить</string>

</property>

</widget>

<widget class="QPushButton" name="refreshButton">

<property name="geometry">

<rect>

<x>280</x>

<y>520</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Обновить</string>

</property>

</widget>

<widget class="QPushButton" name="logoutButton">

<property name="geometry">

<rect>

<x>370</x>

<y>520</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Выйти</string>

</property>

</widget>

</widget>

</widget>

<resources/>

<connections/>

</ui>

};

**Приложение 9**

#ifndef ANALYSTWINDOW\_H

#define ANALYSTWINDOW\_H

#include <QMainWindow>

#include <QTableWidget>

#include "databasemanager.h"

#include <QFileDialog>

#include <QMessageBox>

namespace Ui {

class AnalystWindow;

}

class AnalystWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit AnalystWindow(QWidget \*parent = nullptr);

~AnalystWindow();

private slots:

void loadData();

void exportReport();

private:

Ui::AnalystWindow \*ui;

void setupConnections();

void populateTable(QTableWidget \*table, const QVector<QVector<QString>> &data);

QString generateReport();

};

#endif // ANALYSTWINDOW\_H

**Приложение 10**

#include "analystwindow.h"

#include "ui\_analystwindow.h"

AnalystWindow::AnalystWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::AnalystWindow)

{

ui->setupUi(this);

setupConnections();

loadData();

}

AnalystWindow::~AnalystWindow()

{

delete ui;

}

void AnalystWindow::setupConnections()

{

connect(ui->refreshButton, &QPushButton::clicked, this, &AnalystWindow::loadData);

connect(ui->exportButton, &QPushButton::clicked, this, &AnalystWindow::exportReport);

}

void AnalystWindow::loadData()

{

DatabaseManager &dbManager = DatabaseManager::instance();

auto usersData = dbManager.getUserData();

auto dronesData = dbManager.getDroneData();

auto sectorsData = dbManager.getSectorData();

auto sensorsData = dbManager.getDeviceData();

auto dronePatrolsData = dbManager.getDronePatrols();

auto droneEventsData = dbManager.getDroneEvents();

populateTable(ui->usersTable, usersData);

populateTable(ui->dronesTable, dronesData);

populateTable(ui->sectorsTable, sectorsData);

populateTable(ui->sensorsTable, sensorsData);

populateTable(ui->dronePatrolsTable, dronePatrolsData);

populateTable(ui->droneEventsTable, droneEventsData);

}

void AnalystWindow::populateTable(QTableWidget \*table, const QVector<QVector<QString>> &data)

{

table->setRowCount(data.size());

if (data.isEmpty())

return;

table->setColumnCount(data[0].size());

for (int row = 0; row < data.size(); ++row) {

for (int col = 0; col < data[row].size(); ++col) {

table->setItem(row, col, new QTableWidgetItem(data[row][col]));

}

}

}

void AnalystWindow::exportReport()

{

QString fileName = QFileDialog::getSaveFileName(this, "Save Report", "", "Text Files (\*.txt)");

if (fileName.isEmpty())

return;

QFile file(fileName);

if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QMessageBox::warning(this, "Error", "Failed to open file for writing.");

return;

}

QTextStream out(&file);

out << generateReport();

file.close();

QMessageBox::information(this, "Success", "Report saved successfully.");

}

QString AnalystWindow::generateReport()

{

DatabaseManager &dbManager = DatabaseManager::instance();

QString report;

auto appendSection = [&report](const QString &title, const QVector<QVector<QString>> &data) {

report.append(title + "\n");

for (const auto &row : data) {

report.append(row.join(", ") + "\n");

}

report.append("\n");

};

appendSection("Users:", dbManager.getUserData());

appendSection("Drones:", dbManager.getDroneData());

appendSection("Sectors:", dbManager.getSectorData());

appendSection("Sensors:", dbManager.getDeviceData());

appendSection("Drone Patrols:", dbManager.getDronePatrols());

appendSection("Drone Events:", dbManager.getDroneEvents());

return report;

}

**Приложение 11**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>AnalystWindow</class>

<widget class="QMainWindow" name="AnalystWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>668</height>

</rect>

</property>

<property name="windowTitle">

<string>Analyst Window</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QTableWidget" name="usersTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column>

<property name="text">

<string>ID</string>

</property>

</column>

<column>

<property name="text">

<string>Username</string>

</property>

</column>

<column>

<property name="text">

<string>Role</string>

</property>

</column>

</widget>

</item>

<item>

<widget class="QTableWidget" name="dronesTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column>

<property name="text">

<string>ID</string>

</property>

</column>

<column>

<property name="text">

<string>Name</string>

</property>

</column>

<column>

<property name="text">

<string>Type</string>

</property>

</column>

</widget>

</item>

<item>

<widget class="QTableWidget" name="sectorsTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column>

<property name="text">

<string>ID</string>

</property>

</column>

<column>

<property name="text">

<string>Name</string>

</property>

</column>

<column>

<property name="text">

<string>Description</string>

</property>

</column>

</widget>

</item>

<item>

<widget class="QTableWidget" name="sensorsTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column>

<property name="text">

<string>ID</string>

</property>

</column>

<column>

<property name="text">

<string>Type</string>

</property>

</column>

<column>

<property name="text">

<string>Description</string>

</property>

</column>

</widget>

</item>

<item>

<widget class="QTableWidget" name="dronePatrolsTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>4</number>

</property>

<column>

<property name="text">

<string>DroneID</string>

</property>

</column>

<column>

<property name="text">

<string>SectorID</string>

</property>

</column>

<column>

<property name="text">

<string>TakeoffTimestamp</string>

</property>

</column>

<column>

<property name="text">

<string>TimeEvent</string>

</property>

</column>

</widget>

</item>

<item>

<widget class="QTableWidget" name="droneEventsTable">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column>

<property name="text">

<string>EventID</string>

</property>

</column>

<column>

<property name="text">

<string>DroneID</string>

</property>

</column>

<column>

<property name="text">

<string>TimeEvent</string>

</property>

</column>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QPushButton" name="refreshButton">

<property name="text">

<string>Refresh</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="exportButton">

<property name="text">

<string>Export Report</string>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>26</height>

</rect>

</property>

<widget class="QMenu" name="menuFile">

<property name="title">

<string>File</string>

</property>

<addaction name="actionExit"/>

</widget>

<widget class="QMenu" name="menuHelp">

<property name="title">

<string>Help</string>

</property>

<addaction name="actionAbout"/>

</widget>

<addaction name="menuFile"/>

<addaction name="menuHelp"/>

</widget>

<widget class="QStatusBar" name="statusbar"/>

<action name="actionExit">

<property name="text">

<string>Exit</string>

</property>

</action>

<action name="actionAbout">

<property name="text">

<string>About</string>

</property>

</action>

</widget>

<resources/>

<connections/>

</ui>

**Приложение 12**

#ifndef INFORMATIONHUB\_H

#define INFORMATIONHUB\_H

#include <QWidget>

#include <QTableWidget>

#include "databasemanager.h"

#include "QVBoxLayout"

class InformationHub : public QWidget

{

Q\_OBJECT

public:

explicit InformationHub(QWidget \*parent = nullptr);

private:

QTableWidget \*sectorTableWidget;

QTableWidget \*droneTableWidget;

QTableWidget \*eventsTableWidget;

QTableWidget \*eventTableWidget;

DatabaseManager& databaseManager;

void setupUI();

void populatePatrolSectors();

void populateDroneSensors();

void populateDronePatrols();

void populateDroneEvents();

};

#endif // INFORMATIONHUB\_H

**Приложение 13**

#include "informationhub.h"

InformationHub::InformationHub(QWidget \*parent) : QWidget(parent), databaseManager(DatabaseManager::instance())

{

//qDebug() << "InformationHub constructor called"; // Отладочное сообщение

setupUI();

populatePatrolSectors();

populateDroneSensors();

populateDronePatrols();

populateDroneEvents();

setWindowTitle("Information Hub");

resize(800, 600); // Устанавливаем размер окна

//qDebug() << "InformationHub setup complete"; // Отладочное сообщение

}

void InformationHub::setupUI()

{

//qDebug() << "Setting up UI"; // Отладочное сообщение

QVBoxLayout \*layout = new QVBoxLayout(this);

// Создание таблицы для секторов патрулирования

sectorTableWidget = new QTableWidget(this);

sectorTableWidget->setColumnCount(2);

sectorTableWidget->setHorizontalHeaderLabels(QStringList() << "ID" << "Name");

layout->addWidget(sectorTableWidget);

// Создание таблицы для сенсоров

droneTableWidget = new QTableWidget(this);

droneTableWidget->setColumnCount(4);

droneTableWidget->setHorizontalHeaderLabels(QStringList() << "ID" << "name" << "type" <<"Userid");

layout->addWidget(droneTableWidget);

// Создание таблицы для дронов

eventsTableWidget = new QTableWidget(this);

eventsTableWidget->setColumnCount(3);

eventsTableWidget->setHorizontalHeaderLabels(QStringList() << "ID" << "eventtype" << "eventdescription");

layout->addWidget(eventsTableWidget);

// Создание таблицы для событий

eventTableWidget = new QTableWidget(this);

eventTableWidget->setColumnCount(3);

eventTableWidget->setHorizontalHeaderLabels(QStringList() << "Event ID" << "Drone ID" << "Event Type");

layout->addWidget(eventTableWidget);

setLayout(layout);

//qDebug() << "UI setup complete"; // Отладочное сообщение

}

void InformationHub::populatePatrolSectors()

{

// Получение данных о секторах патрулирования из базы данных

QVector<QVector<QString>> sectorsData = databaseManager.getPatrolSectors();

// Заполнение таблицы секторов патрулирования

for (const auto &sector : sectorsData) {

int row = sectorTableWidget->rowCount();

sectorTableWidget->insertRow(row);

for (int i = 0; i < sector.size(); ++i) {

QTableWidgetItem \*item = new QTableWidgetItem(sector[i]);

sectorTableWidget->setItem(row, i, item);

}

}

}

void InformationHub::populateDroneSensors()

{

// Получение данных о сенсорах из базы данных

QVector<QVector<QString>> droneData = databaseManager.getDroneData();

// Заполнение таблицы сенсоров

for (const auto &drone : droneData) {

int row = droneTableWidget->rowCount();

droneTableWidget->insertRow(row);

for (int i = 0; i < drone.size(); ++i) {

QTableWidgetItem \*item = new QTableWidgetItem(drone[i]);

droneTableWidget->setItem(row, i, item);

}

}

}

void InformationHub::populateDronePatrols()

{

// Получение данных о патрулировании дронов из базы данных

QVector<QVector<QString>> eventData = databaseManager.getEventData();

// Заполнение таблицы патрулирования дронов

for (const auto &event : eventData) {

int row = eventsTableWidget->rowCount();

eventsTableWidget->insertRow(row);

for (int i = 0; i < event.size(); ++i) {

QTableWidgetItem \*item = new QTableWidgetItem(event[i]);

eventsTableWidget->setItem(row, i, item);

}

}

}

void InformationHub::populateDroneEvents()

{

// Получение данных о событиях дронов из базы данных

QVector<QVector<QString>> eventsData = databaseManager.getDroneEvents();

// Заполнение таблицы событий дронов

for (const auto &event : eventsData) {

int row = eventTableWidget->rowCount();

eventTableWidget->insertRow(row);

for (int i = 0; i < event.size(); ++i) {

QTableWidgetItem \*item = new QTableWidgetItem(event[i]);

eventTableWidget->setItem(row, i, item);

}

}

}

**Приложение 14**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>InformationHub</class>

<widget class="QWidget" name="InformationHub">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>400</width>

<height>567</height>

</rect>

</property>

<property name="windowTitle">

<string>Information Hub</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QGroupBox" name="sectorGroupBox">

<property name="title">

<string>Patrol Sectors</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QTableWidget" name="sectorTableWidget">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>2</number>

</property>

<column/>

<column/>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QGroupBox" name="DroneGroupBox">

<property name="title">

<string>Drone</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QTableWidget" name="DroneTableWidget">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>4</number>

</property>

<column/>

<column/>

<column/>

<column/>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QGroupBox" name="eventGroupBox\_2">

<property name="title">

<string>Events</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_5">

<item>

<widget class="QTableWidget" name="eventsTableWidget">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>3</number>

</property>

<column/>

<column/>

<column/>

</widget>

</item>

</layout>

</widget>

</item>

<item>

<widget class="QGroupBox" name="eventGroupBox">

<property name="title">

<string>Drone Events</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout\_5">

<item>

<widget class="QTableWidget" name="eventTableWidget">

<property name="rowCount">

<number>0</number>

</property>

<property name="columnCount">

<number>2</number>

</property>

<column/>

<column/>

</widget>

</item>

</layout>

</widget>

</item>

</layout>

</widget>

<resources/>

<connections/>

</ui>

**Приложение 15**

#ifndef OPERATORWINDOW\_H

#define OPERATORWINDOW\_H

#include <QWidget>

#include <QLabel>

#include <QImage>

#include <QVideoWidget>

#include <QCamera>

#include <QMediaCaptureSession>

#include <QVBoxLayout>

#include <QTextStream>

#include <QMediaDevices>

class OperatorWindow : public QWidget

{

Q\_OBJECT

public:

explicit OperatorWindow(QWidget \*parent = nullptr);

private:

QLabel \*mapLabel;

QVideoWidget \*videoWidget;

QCamera \*camera;

QMediaCaptureSession \*mediaCaptureSession;

void setupUI();

void setupCamera();

void printVideoDevicesInfo();

};

#endif // OPERATORWINDOW\_H

**Приложение 16**

#include "operatorwindow.h"

OperatorWindow::OperatorWindow(QWidget \*parent) : QWidget(parent)

{

setupUI();

setupCamera();

printVideoDevicesInfo();

}

void OperatorWindow::setupUI()

{

QVBoxLayout \*layout = new QVBoxLayout(this);

mapLabel = new QLabel(this);

// Установка изображения карты (путь к файлу или загрузка из ресурсов)

QImage mapImage("C:/Users/DANYA/Desktop/drons/ISOD/ObjectPlane.jpg");

mapLabel->setPixmap(QPixmap::fromImage(mapImage));

mapLabel->setScaledContents(true); // Установка масштабирования изображения

mapLabel->setMaximumSize(400, 200); // Установка максимального размера метки

layout->addWidget(mapLabel);

videoWidget = new QVideoWidget(this);

layout->addWidget(videoWidget);

setLayout(layout);

}

void OperatorWindow::setupCamera()

{

// Получение списка доступных видеоустройств и их характеристик

const QList<QCameraDevice> videoDevices = QMediaDevices::videoInputs();

if (!videoDevices.isEmpty()) {

// Используем первое доступное устройство

const QCameraDevice &device = videoDevices.first();

camera = new QCamera(device);

camera->start();

mediaCaptureSession = new QMediaCaptureSession(this);

mediaCaptureSession->setCamera(camera);

mediaCaptureSession->setVideoOutput(videoWidget);

} else {

// Если устройства не найдены, выдаем сообщение об ошибке

QLabel \*errorLabel = new QLabel("No video devices found", this);

QVBoxLayout \*layout = new QVBoxLayout(this);

layout->addWidget(errorLabel);

setLayout(layout);

}

}

void OperatorWindow::printVideoDevicesInfo()

{

QTextStream out(stdout);

const QList<QCameraDevice> videoDevices = QMediaDevices::videoInputs();

for (const QCameraDevice &device : videoDevices) {

out << "ID: " << device.id() << Qt::endl;

out << "Description: " << device.description() << Qt::endl;

out << "Is default: " << (device.isDefault() ? "Yes" : "No") << Qt::endl;

}

}

**Приложение 17**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>OperatorWindow</class>

<widget class="QWidget" name="OperatorWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>640</width>

<height>480</height>

</rect>

</property>

<property name="windowTitle">

<string>Operator Window</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QLabel" name="mapLabel">

<property name="text">

<string/>

</property>

<property name="scaledContents">

<bool>true</bool>

</property>

</widget>

</item>

<item>

<widget class="QVideoWidget" name="videoWidget" native="true"/>

</item>

</layout>

</widget>

<customwidgets>

<customwidget>

<class>QVideoWidget</class>

<extends>QWidget</extends>

<header>qvideowidget.h</header>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>

**Приложение 18**

#ifndef USERWINDOW\_H

#define USERWINDOW\_H

#include <QWidget>

#include <QComboBox>

#include <QPushButton>

#include <QVBoxLayout>

#include <QSqlQuery>

#include <QDateTime>

#include "databasemanager.h"

#include "informationhub.h"

#include "informationhub.h" // Включим заголовочный файл InformationHub

class UserWindow : public QWidget

{

Q\_OBJECT

public:

explicit UserWindow(QWidget \*parent = nullptr);

~UserWindow() = default;

private slots:

void confirmData();

void changeMode();

void endSession();

void openInformationHub();

private:

void populateSectors();

void populateDrones();

void populateEventTypes();

QComboBox \*sectorComboBox;

QComboBox \*droneComboBox;

QComboBox \*eventTypeComboBox;

QPushButton \*confirmButton;

QPushButton \*changeModeButton;

QPushButton \*endSessionButton;

InformationHub \*informationHub;

};

#endif // USERWINDOW\_H

**Приложение 19**

#include "userwindow.h"

#include "databasemanager.h"

#include "informationhub.h"

#include <QVBoxLayout>

#include <QApplication>

#include <QDebug>

UserWindow::UserWindow(QWidget \*parent) : QWidget(parent), informationHub(nullptr)

{

QVBoxLayout \*layout = new QVBoxLayout(this);

sectorComboBox = new QComboBox(this);

sectorComboBox->setPlaceholderText("Select Patrol Sector");

layout->addWidget(sectorComboBox);

droneComboBox = new QComboBox(this);

droneComboBox->setPlaceholderText("Select Drone");

layout->addWidget(droneComboBox);

eventTypeComboBox = new QComboBox(this);

eventTypeComboBox->setPlaceholderText("Select Event Type");

layout->addWidget(eventTypeComboBox);

confirmButton = new QPushButton("Confirm", this);

layout->addWidget(confirmButton);

connect(confirmButton, &QPushButton::clicked, this, &UserWindow::confirmData);

changeModeButton = new QPushButton("Change Mode", this);

layout->addWidget(changeModeButton);

connect(changeModeButton, &QPushButton::clicked, this, &UserWindow::changeMode);

endSessionButton = new QPushButton("End Session", this);

layout->addWidget(endSessionButton);

connect(endSessionButton, &QPushButton::clicked, this, &UserWindow::endSession);

populateSectors();

populateDrones();

populateEventTypes();

}

void UserWindow::populateSectors()

{

QSqlQuery query(DatabaseManager::instance().m\_database);

query.prepare("SELECT Name FROM PatrolSectors");

if (query.exec()) {

while (query.next()) {

sectorComboBox->addItem(query.value(0).toString());

}

}

}

void UserWindow::populateDrones()

{

QSqlQuery query(DatabaseManager::instance().m\_database);

query.prepare("SELECT Name FROM Drones");

if (query.exec()) {

while (query.next()) {

droneComboBox->addItem(query.value(0).toString());

}

}

}

void UserWindow::populateEventTypes()

{

QSqlQuery query(DatabaseManager::instance().m\_database);

query.prepare("SELECT EventType FROM Events");

if (query.exec()) {

while (query.next()) {

eventTypeComboBox->addItem(query.value(0).toString());

}

}

}

void UserWindow::confirmData()

{

QString selectedSector = sectorComboBox->currentText();

QString selectedDrone = droneComboBox->currentText();

QString selectedEventType = eventTypeComboBox->currentText();

QSqlQuery sectorQuery(DatabaseManager::instance().m\_database);

sectorQuery.prepare("SELECT ID FROM PatrolSectors WHERE Name = :name");

sectorQuery.bindValue(":name", selectedSector);

sectorQuery.exec();

sectorQuery.next();

int sectorId = sectorQuery.value(0).toInt();

QSqlQuery droneQuery(DatabaseManager::instance().m\_database);

droneQuery.prepare("SELECT ID FROM Drones WHERE Name = :name");

droneQuery.bindValue(":name", selectedDrone);

droneQuery.exec();

droneQuery.next();

int droneId = droneQuery.value(0).toInt();

QSqlQuery eventQuery(DatabaseManager::instance().m\_database);

eventQuery.prepare("SELECT ID FROM Events WHERE EventType = :eventType");

eventQuery.bindValue(":eventType", selectedEventType);

eventQuery.exec();

eventQuery.next();

int eventId = eventQuery.value(0).toInt();

QSqlQuery insertQuery(DatabaseManager::instance().m\_database);

insertQuery.prepare("INSERT INTO DroneEvent (EventID, DroneID, TimeEvent) VALUES (:eventID, :droneID, :timeEvent)");

insertQuery.bindValue(":eventID", eventId);

insertQuery.bindValue(":droneID", droneId);

insertQuery.bindValue(":timeEvent", QDateTime::currentDateTime());

if (!insertQuery.exec()) {

qDebug() << "Failed to insert data into DroneEvent:" << insertQuery.lastError();

} else {

qDebug() << "Data inserted successfully into DroneEvent.";

// Закрываем окно InformationHub, если оно открыто

if (informationHub && informationHub->isVisible()) {

informationHub->close();

}

// Открываем заново окно InformationHub

openInformationHub();

}

}

void UserWindow::changeMode()

{

qDebug() << "Change mode button clicked";

openInformationHub();

}

void UserWindow::endSession()

{

qApp->quit();

}

void UserWindow::openInformationHub()

{

qDebug() << "Opening InformationHub";

if (!informationHub) {

informationHub = new InformationHub(nullptr);

}

informationHub->setAttribute(Qt::WA\_DeleteOnClose);

informationHub->setWindowFlag(Qt::Window);

informationHub->resize(800, 600);

informationHub->show();

qDebug() << "InformationHub shown";

}

**Приложение 20**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>UserWindow</class>

<widget class="QWidget" name="UserWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>User Window</string>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QComboBox" name="sectorComboBox">

<property name="currentText">

<string/>

</property>

<property name="placeholderText">

<string>Select Patrol Sector</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="droneComboBox">

<property name="currentText">

<string/>

</property>

<property name="placeholderText">

<string>Select Drone</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="eventTypeComboBox">

<property name="currentText">

<string/>

</property>

<property name="placeholderText">

<string>Select Event Type</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="confirmButton">

<property name="text">

<string>Confirm</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="changeModeButton">

<property name="text">

<string>Change Mode</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="endSessionButton">

<property name="text">

<string>End Session</string>

</property>

</widget>

</item>

</layout>

</widget>

<resources/>

<connections/>

</ui>

**Приложение 21**

#include "login.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Login loginWindow;

loginWindow.show();

return a.exec();

}