ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовой проект
по дисциплине «Объектно-ориентированное программирование»
на тему: **«Создание графического интерфейса с использованием фреймворка Qt»**

Выполнил студент
гр. 3331506/20401 _____ Шарыгин А. В.
(подпись)

Работу принял _____ Ананьевский М. С
(подпись)

Санкт-Петербург
2025 г.

Оглавление

Техническое задание

Необходимо создать графический интерфейс для „умной“ теплицы. С помощью графического интерфейса должно осуществляться взаимодействие с теплицей: чтение данных с датчиков, управление приборами.

1. Введение

Графический интерфейс — важная часть проекта. С его помощью осуществляется удобное взаимодействие с программой. Пользователю намного удобнее управлять объектом через графический интерфейс, вместо того, чтобы разбираться в большом количестве строк кода. Графический интерфейс содержит основные данные, с которыми чаще всего взаимодействует пользователь.

2. Теоретические сведения

В данном проекте для создания графического интерфейса взят фреймворк Qt. Qt — фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

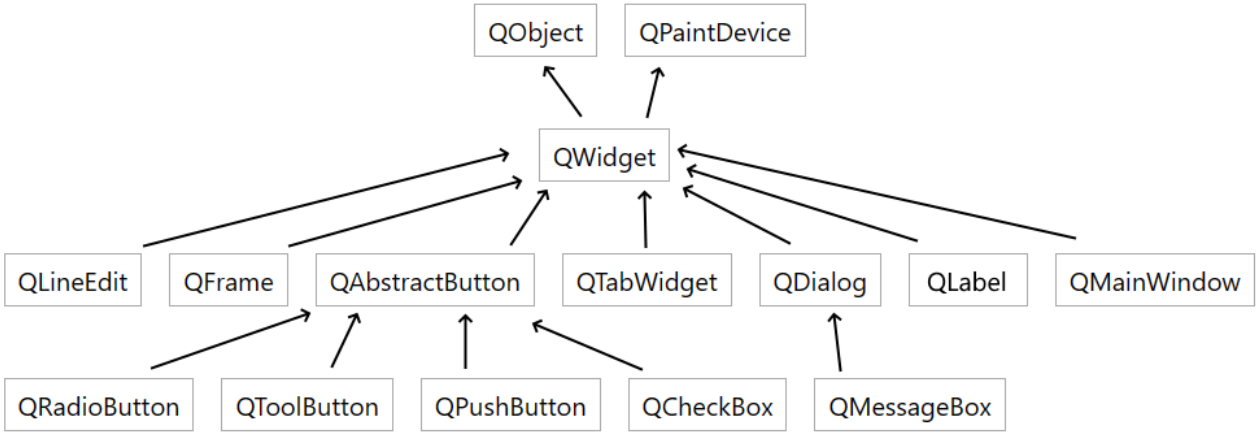
2.1 Qt Creator

Среда разработки Qt Creator не является неотъемлимым компонентом для разработки приложений с помощью фреймворка Qt, тем не менее он упрощает многие вещи, прежде всего конфигурацию и настройку построения приложения. Кроме того, Qt Creator предоставляет унифицированный интерфейс для основных операционных систем.

2.2 Виджеты

Исторически первый подход к построению графического интерфейса на Qt представляли виджет. Виджеты представляют различные элементы пользовательского интерфейса, например, кнопки, текстовые поля и прочие компоненты, из которых состоит окно приложения. Виджет позволяет обрабатывать различные пользовательские события, например, события мыши и клавиатуры. И таким образом пользователь может взаимодействовать с приложением. Базовый встороенный набор виджетов Qt расположен в модуле QtWidgets.

При этом Qt широко использует концепцию наследования. Все виджеты наследуются от встроенного типа QWidget. Это базовый виджет и базовый класс всех виджетов пользовательского интерфейса. Он содержит большинство свойств, необходимых для описания виджета, а также такие свойства позиционирования виджета, цвет и т. д. Иерархию виджетов Qt еще можно представить следующим образом:



2.3 QML и Qt Quick

Изначально использование виджетов из модуля Qt Widgets представляло основной подход к созданию графических приложений на Qt. Однако впоследствии появился второй подход, который представляет специального языка Qt Modeling Language (или сокращенно QML). QML представляет декларативный язык описания пользовательского интерфейса. Он появился вместе с развитием мобильных устройств с сенсорными экранами и позволяет создавать гибкие пользовательские интерфейсы с минимальным написанием кода. Основа функциональности языка QML сосредоточена в одноименном модуле Qt QML, который определяет и реализует язык и его инфраструктуру, а также предоставляет интерфейсы API для интеграции языка QML с JavaScript и C++. Дополнительно модули Qt Quick и Qt Quick Controls предоставляют множество визуальных элементов, анимацию и других компонентов, которые применяются в связке с QML. Таким образом, вместо использования виджетов Qt для проектирования пользовательского интерфейса также можно использовать QML и Qt Quick.

Базовым типом для всех визуальных элементов в Qt Quick является тип Item, который предоставляет общий набор свойств. Фактически он представляет собой прозрачный визуальный элемент, который можно использовать в качестве контейнера. Все остальные визуальные элементы в Qt Quick наследуются от Item. Для создания графического интерфейса в QML модуль Qt Quick Controls

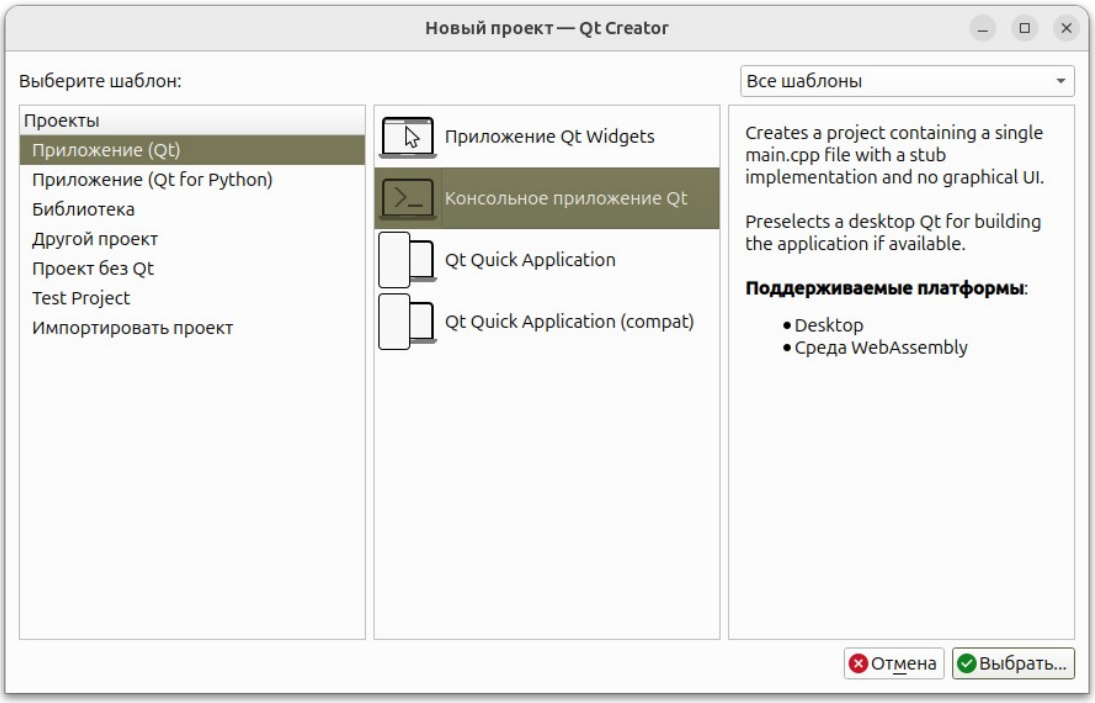
предоставляет набор встроенных компонентов, из которых отмечу основные из них:

- ApplicationWindow: представляет окно верхнего уровня с заголовком и футером
- BusyIndicator: индикатор загрузки
- Button: кнопка
- CheckBox: флажок, который может быть в отмеченном или неотмеченном состоянии
- ComboBox: кнопка со всплывающим окном
- Dial: компонент в виде кругового набора (как на старых стационарных телефонах)
- Dialog: диалоговое окно
- Label: метка с текстом
- Popup: высплывающее окно
- ProgressBar: индикатор прогресса операции
- RadioButton: радиокнопка или переключатель
- ScrollBar: вертикальные и горизонтальные полосы прокрутки
- ScrollView: визуальный компонент, который поддерживает прокрутку
- Slider: слайдер для выбора числового значения из некоторого диапазона
- SpinBox: выпадающий список
- Switch: кнопка-переключатель
- TextArea: элемент для ввода многострочного текста
- TextField: элемент для ввода однострочного текста
- ToolTip: всплывающая подсказка
- Tumbler: прокручиваемый список элементов для выбора

3. Ход работы

3.1 Создание проекта в Qt Creator

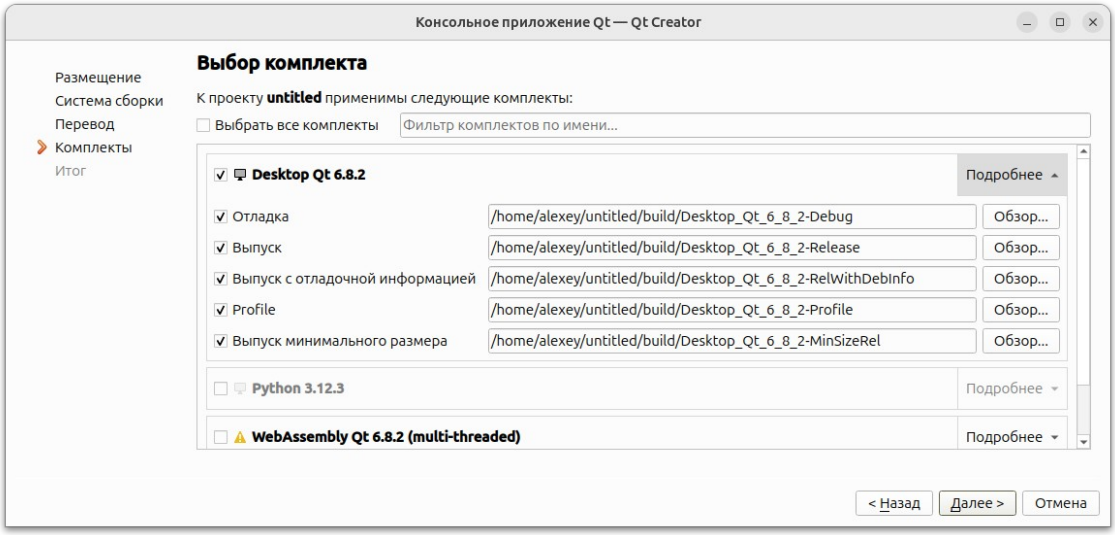
1) Создадим новый проект в Qt Creator, выберем **консольное приложение Qt**:



2) Далее назовем проект **smart_farm_app** и выберем размещение.

3) Выберем систему сборки CMake.

4) На шаге **Выбор комплекта** сделаем так:



3.2 Редактируем файл CmakeLists.txt

```
# Устанавливает минимально требуемую версию CMake (3.16)
cmake_minimum_required(VERSION 3.16)

# Определяет проект с именем "smart_farm_app" и указывает, что используется язык C++
project(smart_farm_app LANGUAGES CXX)

# Указывает, что стандарт C++ обязателен (без fallback на старые версии)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Ищет и подключает необходимые компоненты Qt6: Core, Quick и SerialPort
# Пакет должен быть найден, иначе конфигурация завершится ошибкой (REQUIRED)
find_package(Qt6 REQUIRED COMPONENTS Core Quick SerialPort)

# Настраивает стандартные параметры проекта для Qt (например, автоматическое подключение MOC)
qt_standard_project_setup()

# Создает исполняемый файл smart_farm_app из указанных исходных файлов (main.cpp)
# Специальная Qt-версия add_executable, которая добавляет необходимую обработку для Qt
qt_add_executable(smart_farm_app
  main.cpp
)

# Добавляет QML-модуль к проекту:
# URI path - пространство имен для QML-типов
# VERSION 1.0 - версия модуля
# QML_FILES - список QML-файлов (main.qml)
# SOURCES - C++ исходники, которые могут использоваться из QML
qt_add_qml_module(smart_farm_app
    URI path
    VERSION 1.0
    QML_FILES main.qml   // добавиться автоматически позже
```

```
    SOURCES sensors_handler.h sensors_handler.cpp  // добавиться автоматически позже
)

# Подключает необходимые библиотеки Qt к целевому исполняемому файлу
target_link_libraries(smart_farm_app PRIVATE Qt6::Core Qt6::Quick Qt6::SerialPort)

# Включает модуль GNUInstallDirs, который предоставляет стандартные пути установки
include(GNUInstallDirs)

# Устанавливает правила инсталляции для целевого файла smart_farm_app:
# LIBRARY - устанавливает библиотеки в стандартную директорию для библиотек
# RUNTIME - устанавливает исполняемые файлы в стандартную директорию для бинарников
install(TARGETS smart_farm_app
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

3.3 Добавим класс sensors_handler

3.3.1 Редактируем файл sensors_handler.h

```
// Защита от повторного включения заголовочного файла
#ifndef SENSORS_HANDLER_H
#define SENSORS_HANDLER_H

// Подключаем необходимые заголовочные файлы Qt
#include <QObject>       // Базовый класс для объектов Qt
#include <QSerialPort>   // Класс для работы с последовательным портом
#include <QByteArray>    // Класс для работы с бинарными данными

// Класс для работы с датчиками и светодиодами, наследуется от QObject
class Sensors_handler : public QObject
{
```

```
    Q_OBJECT  // Макрос Qt для поддержки сигналов/слотов и системы мета-объектов


    // Свойства Qt, доступные из QML:
    Q_PROPERTY(int time READ get_time NOTIFY sensorValuesChanged)          // Время измерения
    Q_PROPERTY(int temperature READ get_temperature NOTIFY sensorValuesChanged) // Температура
    Q_PROPERTY(int humidity READ get_humidity NOTIFY sensorValuesChanged)       // Влажность воздуха
    Q_PROPERTY(int humidity_dirt READ get_humidity_dirt NOTIFY sensorValuesChanged) // Влажность почвы


public:
    // Конструктор с необязательным указанием родительского объекта
    explicit Sensors_handler(QObject *parent = nullptr);


    // Деструктор
    ~Sensors_handler();


    // Методы для чтения значений датчиков (const означает, что они не изменяют состояние объекта)
    int get_time() const;            // Получить время
    int get_temperature() const;     // Получить температуру
    int get_humidity() const;        // Получить влажность воздуха
    int get_humidity_dirt() const;   // Получить влажность почвы


    // Методы, которые можно вызывать из QML (благодаря Q_INVOKABLE)
    Q_INVOKABLE void startReading(const QString &portName);  // Начать чтение с указанного порта
    Q_INVOKABLE void sendCommand(const QString &command);    // Отправить команду через порт


signals:
    // Сигнал, который генерируется при изменении значений датчиков
    void sensorValuesChanged();


private slots:
```

```
    // Слот для обработки входящих данных с последовательного порта
    void readData();

private:
    // Приватные члены класса:
    QSerialPort *serial;     // Объект для работы с последовательным портом
    QByteArray buffer;       // Буфер для накопления входящих данных
    int m_time;              // Время измерения
    int m_temperature;       // Температура
    int m_humidity;          // Влажность воздуха (возможна опечатка, должно
быть m_humidity)
    int m_humidity_dirt;     // Влажность почвы
};

// Завершение защиты от повторного включения
#endif // SENSORS_HANDLER_H
```

3.3.2 Редактируем файл sensors_handler.cpp

```
// Подключение заголовочного файла класса
#include "sensors_handler.h"
// Подключение модуля для вывода отладочной информации
#include <QDebug>

// Конструктор класса:
// - Инициализирует родительский QObject
// - Создает объект QSerialPort
// - Инициализирует переменные-члены нулевыми значениями
// - Устанавливает соединение сигнала readyRead с обработчиком readData
Sensors_handler::Sensors_handler(QObject *parent)
    : QObject(parent), serial(new QSerialPort(this)), m_temperature(0),
m_humidity(0), m_humidity_dirt(0), m_time(0)
{
    connect(serial, &QSerialPort::readyRead, this, &Sensors_handler::readData);
}
```

```
// Деструктор класса:
// - Закрывает последовательный порт, если он открыт
Sensors_handler::~Sensors_handler()
{
    if(serial->isOpen()) {
        serial->close();
    }
}

// Геттер для температуры
int Sensors_handler::get_temperature() const
{
    return m_temperature;
}

// Геттер для влажности воздуха
int Sensors_handler::get_humidity() const
{
    return m_humidity;
}

// Геттер для влажности почвы
int Sensors_handler::get_humidity_dirt() const
{
    return m_humidity_dirt;
}

// Геттер для времени (предположительно, времени снятия показаний)
int Sensors_handler::get_time() const
{
    return m_time;
}

// Метод для начала чтения данных с последовательного порта:
// - Устанавливает параметры порта (имя, скорость, биты данных и т.д.)
```

```
// - Пытается открыть порт и выводит сообщение об успехе/ошибке
void Sensors_handler::startReading(const QString &portName)
{
    serial->setPortName(portName);
    serial->setBaudRate(QSerialPort::Baud115200);
    serial->setDataBits(QSerialPort::Data8);
    serial->setParity(QSerialPort::NoParity);
    serial->setStopBits(QSerialPort::OneStop);
    serial->setFlowControl(QSerialPort::NoFlowControl);

    if (serial->open(QIODevice::ReadWrite)) {
        qDebug() << "Порт успешно открыт!";
    } else {
        qDebug() << "Ошибка открытия порта:" << serial->errorString();
    }
}

// Метод для отправки команды через последовательный порт:
// - Проверяет, открыт ли порт
// - Преобразует команду в UTF-8 и добавляет символ новой строки
// - Отправляет данные и выводит отладочное сообщение
void Sensors_handler::sendCommand(const QString &command)
{
    if(serial->isOpen()) {
        QByteArray data = command.toUtf8() + "\n";
        serial->write(data);
        qDebug() << "Отправлена команда" << command;
    }
    else {
        qDebug() << "Порт не открыт, команда не отправлена.";
    }
}

// Метод для чтения данных из последовательного порта:
// - Добавляет новые данные в буфер
```

```
// - Ищет символ новой строки как признак конца сообщения
// - Разбирает строку на отдельные значения (разделенные запятыми)
// - Обновляет переменные-члены и генерирует сигнал при успешном разборе
void Sensors_handler::readData()
{
    buffer += serial->readAll();

    int pos = buffer.indexOf('\n');
    if(pos != -1) {
        QByteArray line = buffer.left(pos).trimmed();
        buffer = buffer.mid(pos + 1);

        qDebug() << "Полученная строка: " << line;

        QList<QByteArray> values = line.split(',');
        if(values.size() == 4) {
            m_time = values[0].toInt();
            m_temperature = values[1].toInt();
            m_humidity = values[2].toInt();
            m_humidity_dirt = values[3].toInt();
            emit sensorValuesChanged();
        }
        else {
            qDebug() << "Ошибка: получено неверное количество значений.";
        }
    }
}
```

3.4 Редактируем файл main.cpp

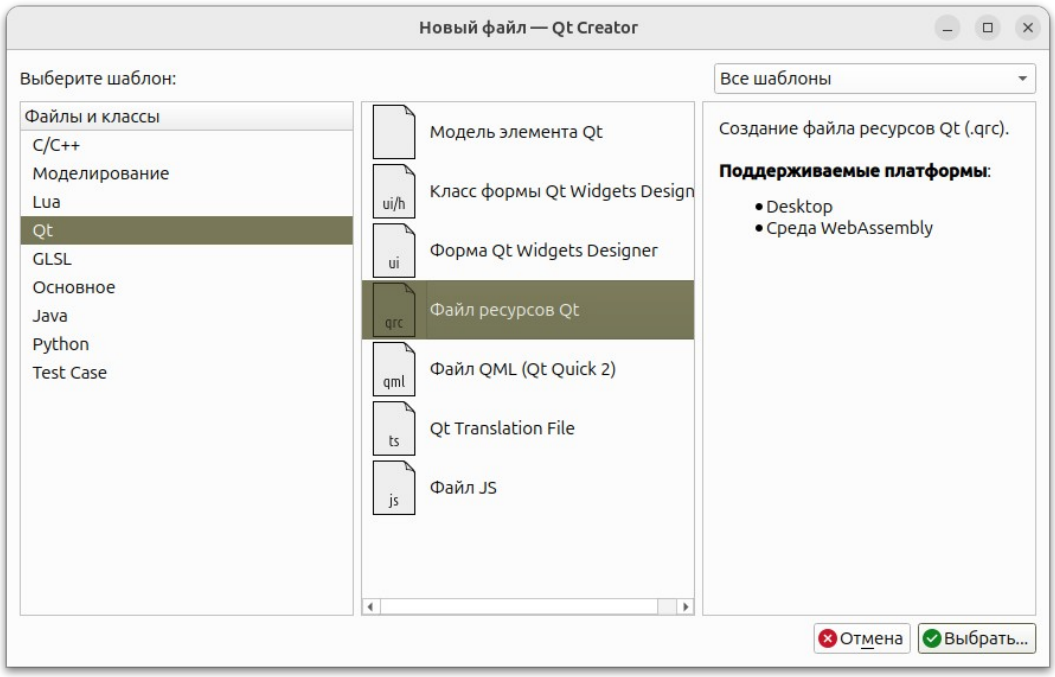
```
// Подключаем необходимые заголовочные файлы Qt
#include <QGuiApplication>         // Базовый класс GUI-приложения (без
QWidgets)
#include <QQmlApplicationEngine>   // Движок для работы с QML
#include "sensors_handler.h"  // Наш собственный класс для работы с датчиками
```

```
// Точка входа в приложение
int main(int argc, char *argv[])
{
    // Создаем объект приложения Qt (основной цикл обработки событий)
    QGuiApplication app(argc, argv);

    // Создаем QML-движок, который будет загружать и выполнять QML-код
    QQmlApplicationEngine engine;

    // Регистрируем наш C++ класс в QML системе:
    // - "sense" - имя модуля (будет использоваться в QML import)
    // - 1, 0 - версия модуля (1.0)
    // - "Sensors_handler" - имя типа в QML
    qmlRegisterType<Sensors_handler>("sense", 1, 0, "Sensors_handler");

    // Указываем URL главного QML-файла (используется qrc-система ресурсов Qt)
    const QUrl url("qrc:/path/main.qml");

    // Загружаем QML-файл в движок
    engine.load(url);

    // Запускаем главный цикл обработки событий приложения
    // (приложение будет работать, пока не будет закрыто)
    return app.exec();
}
```

3.5 Файл main.qml

3.5.1 Создание файла main.qml

Создадим файл main.qml в нашем проекте:



3.5.2 Редактируем файл main.qml

```
// Импорт необходимых модулей QtQuick
import QtQuick          // Базовые QML-типы
import QtQuick.Controls // UI-элементы (кнопки, переключатели и т.д.)
import sense 1.0        // Наш зарегистрированный C++ модуль с датчиками

// Главное окно приложения
ApplicationWindow {
    width: 300      // Ширина окна
    height: 500     // Высота окна
    visible: true   // Делаем окно видимым
    title: "Smart_farm"  // Заголовок окна

    // Создаем экземпляр нашего C++ класса для работы с датчиками
    Sensors_handler {
        id: sensors_handler  // Идентификатор для доступа из QML
```

```
    // Обработчик сигнала об изменении значений датчиков
    onSensorValuesChanged: {
        text_time.text = "Текущее время: " + time
        text_temperature.text = "Температура: " + temperature
        text_humidity.text = "Влажность воздуха: " + humidity
        text_humidity_dirt.text = "Влажность почвы: " + humidity_dirt
    }
}

// Вертикальная компоновка элементов
Column {
    spacing: 10             // Расстояние между элементами
    anchors.centerIn: parent // Центрирование в родительском элементе

    // Текстовое поле для отображения времени
    Text {
        id: text_time
        text: "Текущее время: -"  // Начальное значение
    }

    // Текстовое поле для отображения температуры
    Text {
        id: text_temperature
        text: "Температура: -"
    }

    // Текстовое поле для отображения влажности воздуха
    Text {
        id: text_humidity
        text: "Влажность воздуха: -"
    }

    // Текстовое поле для отображения влажности почвы
    Text {
```

```
    id: text_humidity_dirt
    text: "Влажность почвы: -"
}

// Кнопка для начала чтения данных с датчиков
Button {
    text: "Начать чтение"
    // При клике вызываем метод C++ класса с указанием порта
    onClicked: sensors_handler.startReading("ttyACM0")
}

// Переключатель для управления лампой
Switch {
    text: "Лампа"
    checked: false  // Начальное состояние - выключено
    // При изменении состояния отправляем соответствующую команду
    onToggled: {
        if(checked === true) {
            sensors_handler.sendCommand("l")  // Включить
        } else {
            sensors_handler.sendCommand("k")  // Выключить
        }
    }
}

// Переключатель для управления нагревателем
Switch {
    text: "Нагреватель"
    checked: false
    onToggled: {
        if(checked === true) {
            sensors_handler.sendCommand("h")
        } else {
            sensors_handler.sendCommand("j")
        }
```

```
            }
        }

        // Переключатель для управления насосом
        Switch {
            text: "Насос"
            checked: false
            onToggled: {
                if(checked === true) {
                    sensors_handler.sendCommand("p")
                } else {
                    sensors_handler.sendCommand("o")
                }
            }
        }

        // Переключатель для управления вентилятором
        Switch {
            text: "Вентилятор"
            checked: false
            onToggled: {
                if(checked === true) {
                    sensors_handler.sendCommand("v")
                } else {
                    sensors_handler.sendCommand("b")
                }
            }
        }
    }
}
```

3.6 Код для Arduino

```
#include <DHT11.h>

#define DHTPIN 12 // влажность и температура воздуха
#define LAMP_PIN 6 // лампа
#define FAN_PIN 7 // вентилятор
#define NAGREV_PIN 4 // наргреватель
#define PUMP_PIN 5 // помпа
#define HUM_SOIL_PIN A1 // влажность почвы

DHT11 dht11(DHTPIN);

class Climate {
public:
int temperature_min;
int temperature_max;
int humidity_dirt_min;
int humidity_dirt_max;
};

class Datatime {
private:
int hour;
public:
Datatime(int current_time);
int get_time();
void set_time(int new_time);
void get_time_from_serial();
};

class Thermometer {
private:
int temperature;
public:
int get_temperature();
```

```
void get_temperature_from_thermometer();
};

class Heater {
public:
bool on_temperature;
bool on_manual_control;
long long int time_off;

Heater();
void start();
void stop();
};

class Humidity_sensor {
private:
int humidity;
public:
Humidity_sensor();
bool good;
void check_dood();
int get_humidity();
void get_humidity_from_sensor();
};

class Ventilator {
public:
bool on_temperature;
bool on_schedule;
bool on_after_off_heater;
bool on_manual_control;

Ventilator();
void start();
void stop();
```

```
};

class Humidity_dirt_sensor {
private:
int humidity_dirt;
public:
int get_humidity_dirt();
void get_humidity_dirt_from_sensor();
};

class Pump {
public:
bool on_humidity_dirt;
bool on_manual_control;
unsigned long int time_on;
unsigned long int time_off;

Pump();

void start();
void stop();
};

class Lamp {
public:
bool on_schedule;
bool on_manual_control;

Lamp();

void start();
void stop();
};
```

```
int Thermometer::get_temperature()
{
return temperature;
}

void Thermometer::get_temperature_from_thermometer()
{
temperature = dht11.readTemperature();
}

Humidity_sensor::Humidity_sensor()
{
good = true;
}

int Humidity_sensor::get_humidity()
{
return humidity;
}

void Humidity_sensor::get_humidity_from_sensor()
{
humidity = dht11.readHumidity();
}

int Humidity_dirt_sensor::get_humidity_dirt() {
return humidity_dirt;
}

void Humidity_dirt_sensor::get_humidity_dirt_from_sensor()
{
humidity_dirt = analogRead(HUM_SOIL_PIN);
}

Datatime::Datatime(int current_time)
```

```
{
hour = current_time;
}

int Datatime::get_time()
{
int time = hour + millis() / 3600000;
return time % 24;
}

void Datatime::set_time(int new_time)
{
hour = new_time;
}

Heater::Heater()
{
on_temperature = false;
on_manual_control = false;
}

void Heater::start()
{
digitalWrite(NAGREV_PIN, true);
}

void Heater::stop()
{
digitalWrite(NAGREV_PIN, false);
}

Ventilator::Ventilator()
{
on_schedule = false;
on_temperature = false;
```

```
on_after_off_heater = false;
on_manual_control = false;
}

void Ventilator::start()
{
digitalWrite(FAN_PIN, true);
}

void Ventilator::stop()
{
digitalWrite(FAN_PIN, false);
}

Lamp::Lamp()
{
on_schedule = false;
on_manual_control = false;
}

void Lamp::start()
{
digitalWrite(LAMP_PIN, true);
}

void Lamp::stop()
{
digitalWrite(LAMP_PIN, false);
}

Pump::Pump()
{
on_humidity_dirt = false;
on_manual_control = false;
time_on = 0;
```

```
time_off = 0;
}

void Pump::start()
{
digitalWrite(PUMP_PIN, true);
}

void Pump::stop()
{
digitalWrite(PUMP_PIN, false);
}

void set_climate(Climate &climate)
{
climate.temperature_max = 24;
climate.temperature_min = 20;
climate.humidity_dirt_max = 10;
climate.humidity_dirt_min = 5;
}

void control_temperature(const Climate &climate,
                         const Thermometer &thermometer,
                         Heater &heater,
                         Ventilator &ventilator)
{
if(thermometer.get_temperature() < climate.temperature_min) {
heater.on_temperature = true;
ventilator.on_temperature = true;
ventilator.on_after_off_heater = false;
      }
      else if(thermometer.get_temperature() > climate.temperature_max) {
            if(heater.on_temperature) {
                  ventilator.on_after_off_heater = true;
                  heater.time_off = millis();
```

```
            }
            heater.on_temperature = false;
            ventilator.on_temperature = false;
      }
}

void control_ventilation(const Datatime &datatime, Ventilator &ventilator)
{
      if(datatime.get_time() < 6 || datatime.get_time() > 23) {
            ventilator.on_schedule = false;
            return;
      }

      if(datatime.get_time() % 4 == 0) {
            ventilator.on_schedule = true;
      }
      else {
            ventilator.on_schedule = false;
      }
}

void control_lighting(const Datatime &datatime, Lamp &lamp)
{
      if(datatime.get_time() > 17 && datatime.get_time() < 22) {
      lamp.on_schedule = true;
      }
      else {
      lamp.on_schedule = false;
      }
}

void control_watering(const Climate &climate, const Humidity_dirt_sensor
&humidity_dirt_sensor, Pump &pump, int time_wait, int time_watering)
{
      if((humidity_dirt_sensor.get_humidity_dirt() < climate.humidity_dirt_max)
      && (millis() - pump.time_off > time_wait)) {
```

```
            if(pump.on_humidity_dirt == false) {
                  pump.time_on = millis();
            }
            pump.on_humidity_dirt = true;
      }

      if(humidity_dirt_sensor.get_humidity_dirt() > climate.humidity_dirt_max) {
      pump.on_humidity_dirt = false;
            return;
      }

      if(millis() - pump.time_on > time_watering) {
            if(pump.on_humidity_dirt == true) {
            pump.time_off = millis();
            }
      pump.on_humidity_dirt = false;
      }
}

void control_ventilation_after_off_heater(Ventilator &ventilator, Heater
&heater, int time_ventilation)
{
      if(ventilator.on_after_off_heater && (millis() - heater.time_off >
      time_ventilation)) {
            ventilator.on_after_off_heater = false;
      }
}

void do_heat(const Heater &heater)
{
      if(heater.on_temperature || heater.on_manual_control) {
            heater.start();
      }
      else {
            heater.stop();
      }
```

```
}

void do_ventilation(const Ventilator &ventilator)
{
	if(ventilator.on_schedule || ventilator.on_temperature
	|| ventilator.on_after_off_heater || ventilator.on_manual_control) {
		ventilator.start();
	}
	else {
		ventilator.stop();
	}
}

void do_lighting(const Lamp &lamp)
{
	if(lamp.on_schedule || lamp.on_manual_control) {
		lamp.start();
	}
	else {
		lamp.stop();
	}
}

void do_watering(const Pump &pump, const Heater &heater)
{
	if((pump.on_humidity_dirt || pump.on_manual_control) &&
	heater.on_temperature == false) {
		pump.start();
	}
	else {
		pump.stop();
	}
}
```

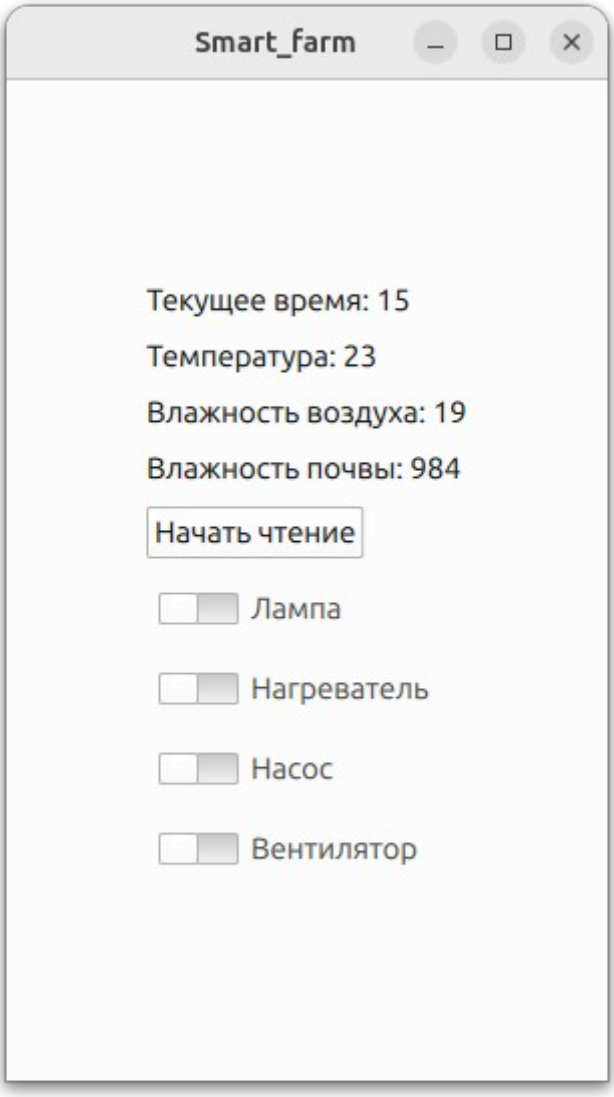
```
void parse(Lamp &lamp, Ventilator &ventilator, Heater &heater, Pump &pump)
{
if(Serial.available() > 1) {
      char key = Serial.read();
      switch(key) {
            case 'l':
                  lamp.on_manual_control = true;
                  while(Serial.available() > 0) Serial.read();
                  break;
            case 'k':
                  lamp.on_manual_control = false;
                  while(Serial.available() > 0) Serial.read();
                  break;
            case 'v':
                  ventilator.on_manual_control = true;
                  while(Serial.available() > 0) Serial.read();
                  break;
            case 'b':
                  ventilator.on_manual_control = false;
                  while(Serial.available() > 0) Serial.read();
                  break;

            case 'h':
                  heater.on_manual_control = true;
                  while(Serial.available() > 0) Serial.read();
                  break;
            case 'j':
                  heater.on_manual_control = false;
                  while(Serial.available() > 0) Serial.read();
                  break;

            case 'p':
                  pump.on_manual_control = true;
                  while(Serial.available() > 0) Serial.read();
                  break;
```

```
            case 'o':
                  pump.on_manual_control = false;
                  while(Serial.available() > 0) Serial.read();
                  break;
            default:
                  while(Serial.available() > 0) Serial.read();
                  Serial.println("Не правильная команда!");
            }
      }
}

Climate climate_1;
Datatime datatime(13);

Thermometer thermometer_1;
Humidity_sensor humidity_sensor_1;
Humidity_dirt_sensor humidity_dirt_sensor_1;

Heater heater_1;
Ventilator ventilator_1;
Lamp lamp_1;
Pump pump_1;

void setup() {
Serial.begin(115200);

pinMode(LAMP_PIN, OUTPUT); // лампа
pinMode(FAN_PIN, OUTPUT); // вентилятор
pinMode(NAGREV_PIN, OUTPUT); // нагреватель
pinMode(PUMP_PIN, OUTPUT); // помпа

set_climate(climate_1);
}

void loop() {
```

```
static unsigned long lastReadTime = 0;
if(millis() - lastReadTime > 100) {
thermometer_1.get_temperature_from_thermometer();
humidity_sensor_1.get_humidity_from_sensor();
humidity_dirt_sensor_1.get_humidity_dirt_from_sensor();
}
parse(lamp_1, ventilator_1, heater_1, pump_1);

Serial.print(datatime.get_time());
Serial.print(",");
Serial.print(thermometer_1.get_temperature());
Serial.print(",");
Serial.print(humidity_sensor_1.get_humidity());
Serial.print(",");
Serial.println(humidity_dirt_sensor_1.get_humidity_dirt());

control_temperature(climate_1, thermometer_1, heater_1, ventilator_1);
control_ventilation(datatime, ventilator_1);
control_lighting(datatime, lamp_1);
control_watering(climate_1, humidity_dirt_sensor_1, pump_1, 6000, 3000);
control_ventilation_after_off_heater(ventilator_1, heater_1, 2000);

do_heat(heater_1);
do_ventilation(ventilator_1);
do_lighting(lamp_1);
do_watering(pump_1, heater_1);
}
```

4. Результат работы

В результате работы появляется окно **Smart_farm**, в котором отображаются: время, данные климата и переключатели приборов. Любой из приборов можно включить или выключить.



5. Заключение

В ходе выполнения курсового проекта были изучены возможности фреймворка Qt. Было создано приложение для удобного взаимодействия с „умной“ теплицей.

6. Литература

1. https://metanit.com/cpp/qt/