ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИМЕНИ И.М. ГУБКИНА»

ФАКУЛЬТЕТ КОМПЛЕКСНОЙ БЕЗОПАСНОСТИ ТЭК
КАФЕДРА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №2

по дисциплине «Специализированные языки и технологии программирования»

на тему «Работа с базовыми типами, реализация сигналов и слотов»

Выполнил студент:
группы КА-22-06
Воронин Алексей Дмитриевич

Преподаватель:
Греков Владимир Сергеевич

Москва, 2025

Оглавление

Цель работы:

- Освоение механизма сигналов и слотов в Qt.
- Практика работы с базовыми типами данных Qt.

Задания:

- Изучить базовые операции с QString.
- Реализовать простую форму для ввода текста и кнопку для его обработки.
- Использовать сигналы и слоты для реализации логики приложения.

ЧАСТЬ 1. Работа с QString и сигналами/слотами

Шаг 1. Создание проекта и формы.

Запустите Qt Creator и создайте новый проект "Приложение Qt Widgets".

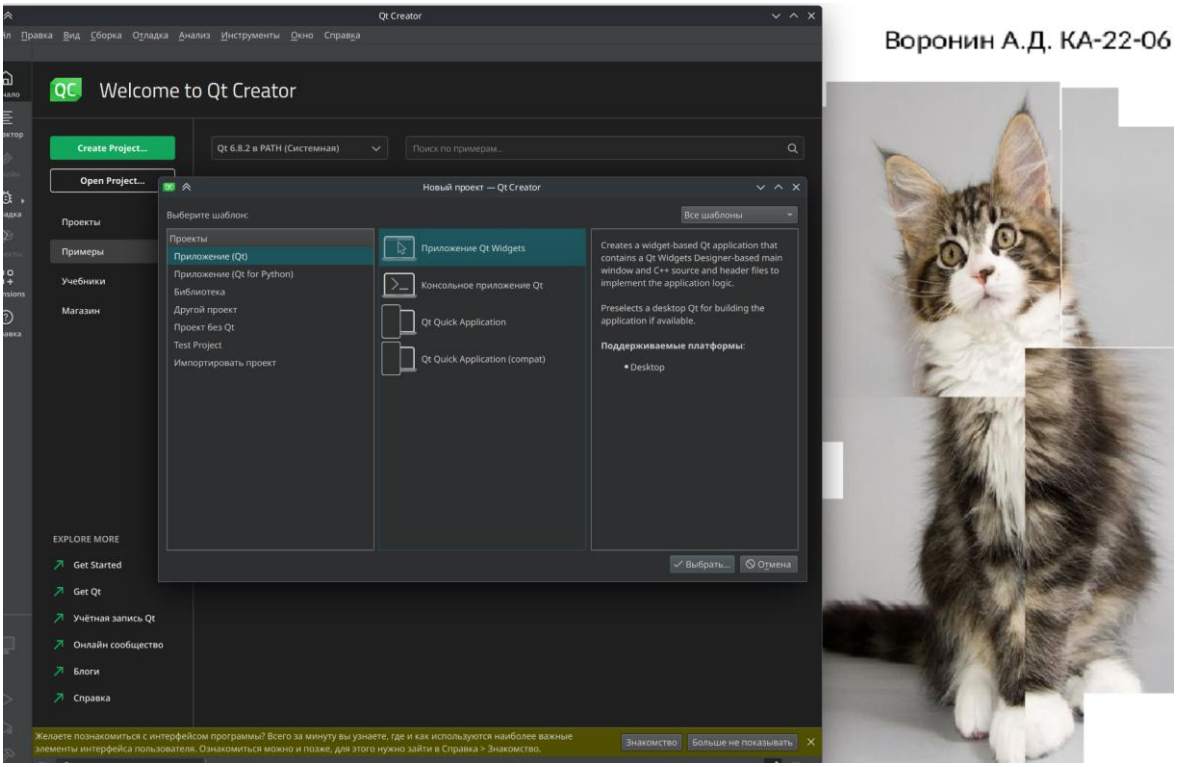

Рисунок 1 – Создание приложения

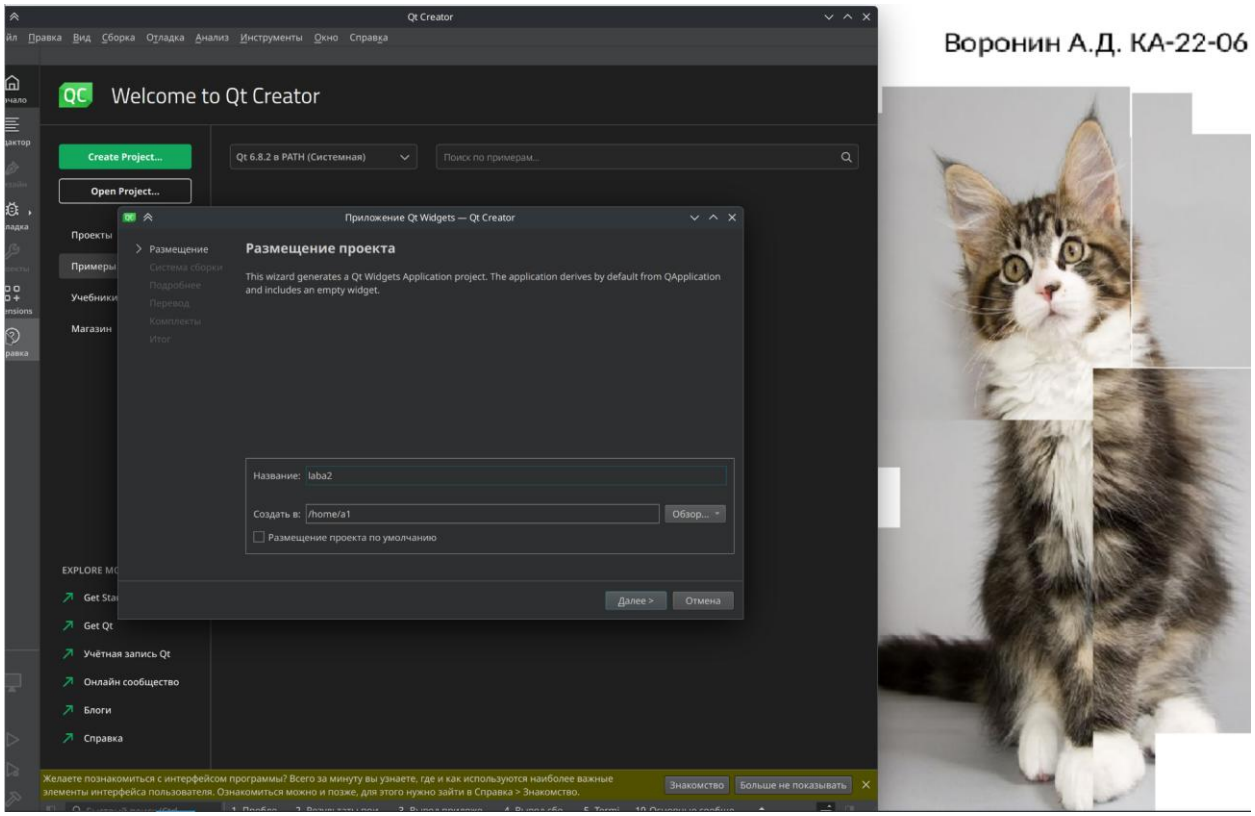

Рисунок 2 – Создание приложения

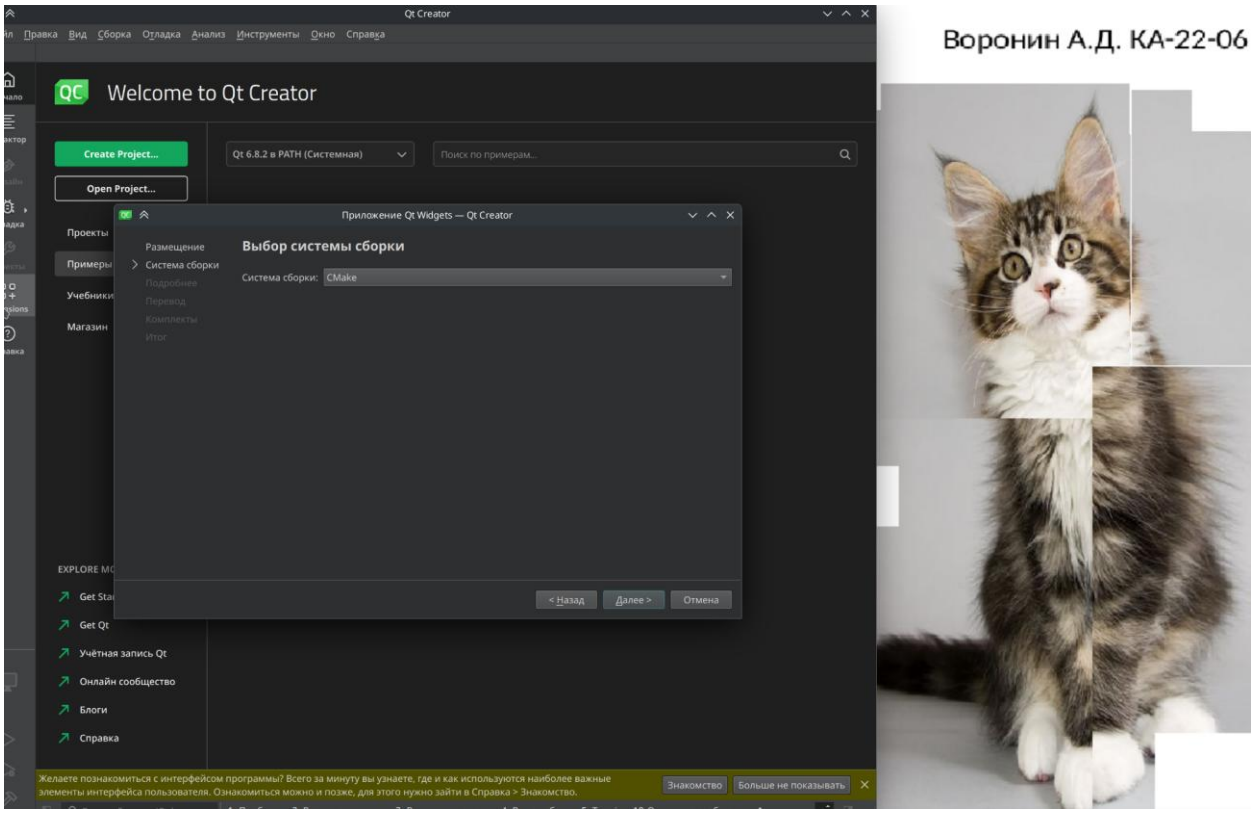

Рисунок 3 – Создание приложения

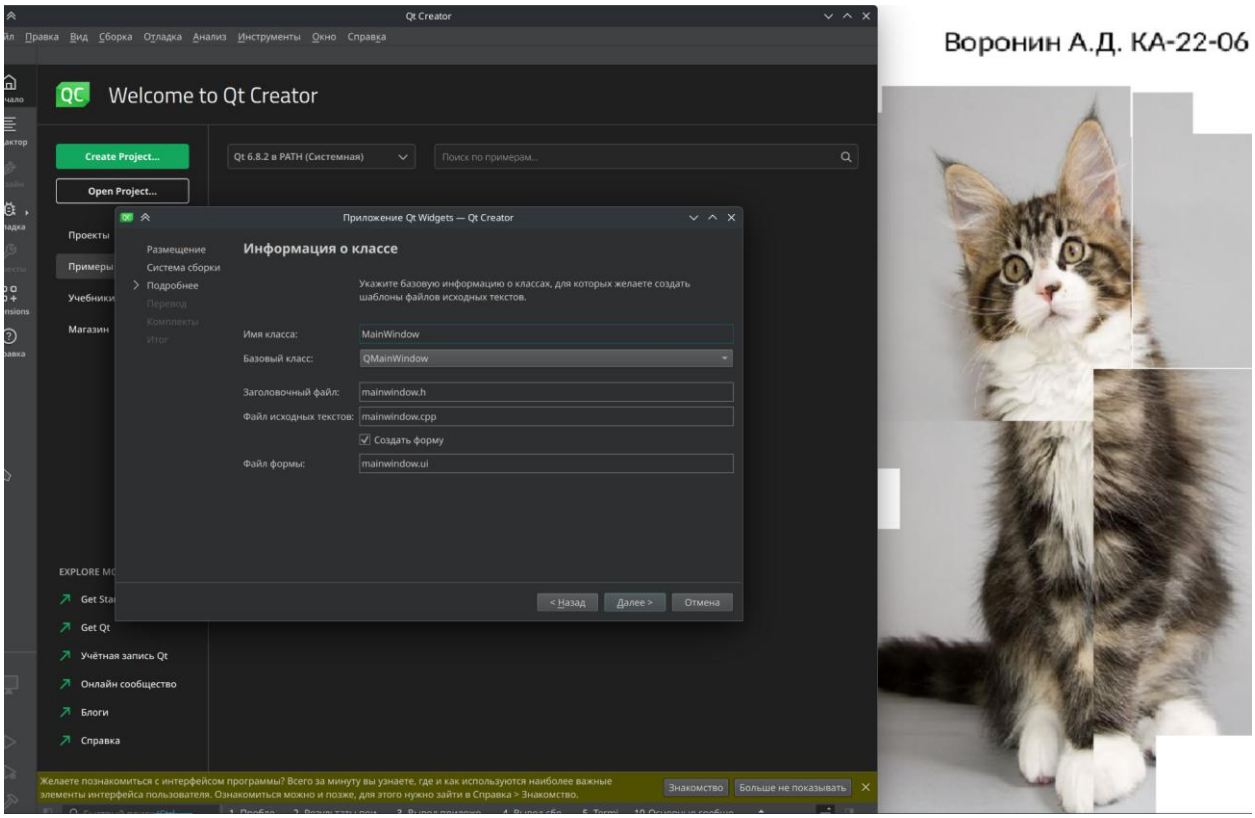

Рисунок 4 – Создание приложения

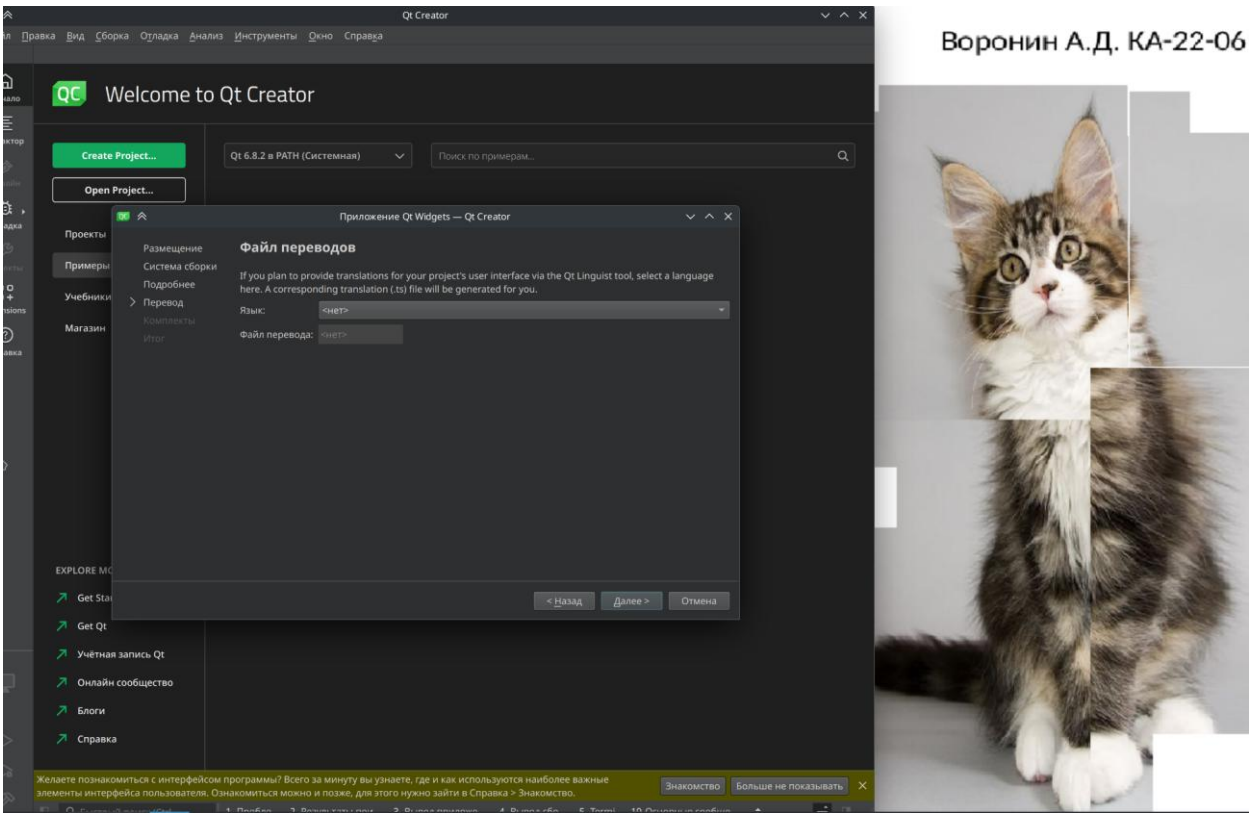

Рисунок 5 – Создание приложения

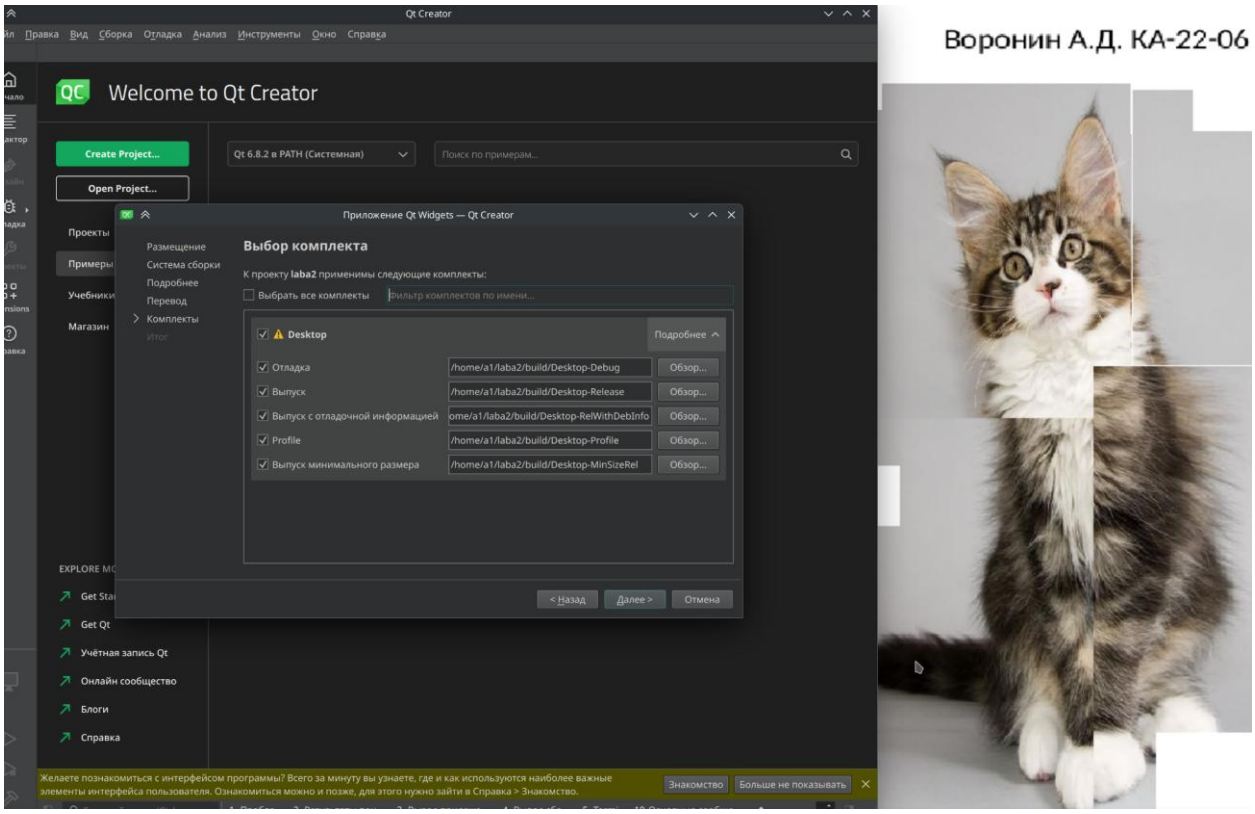

Рисунок 6 – Создание приложения

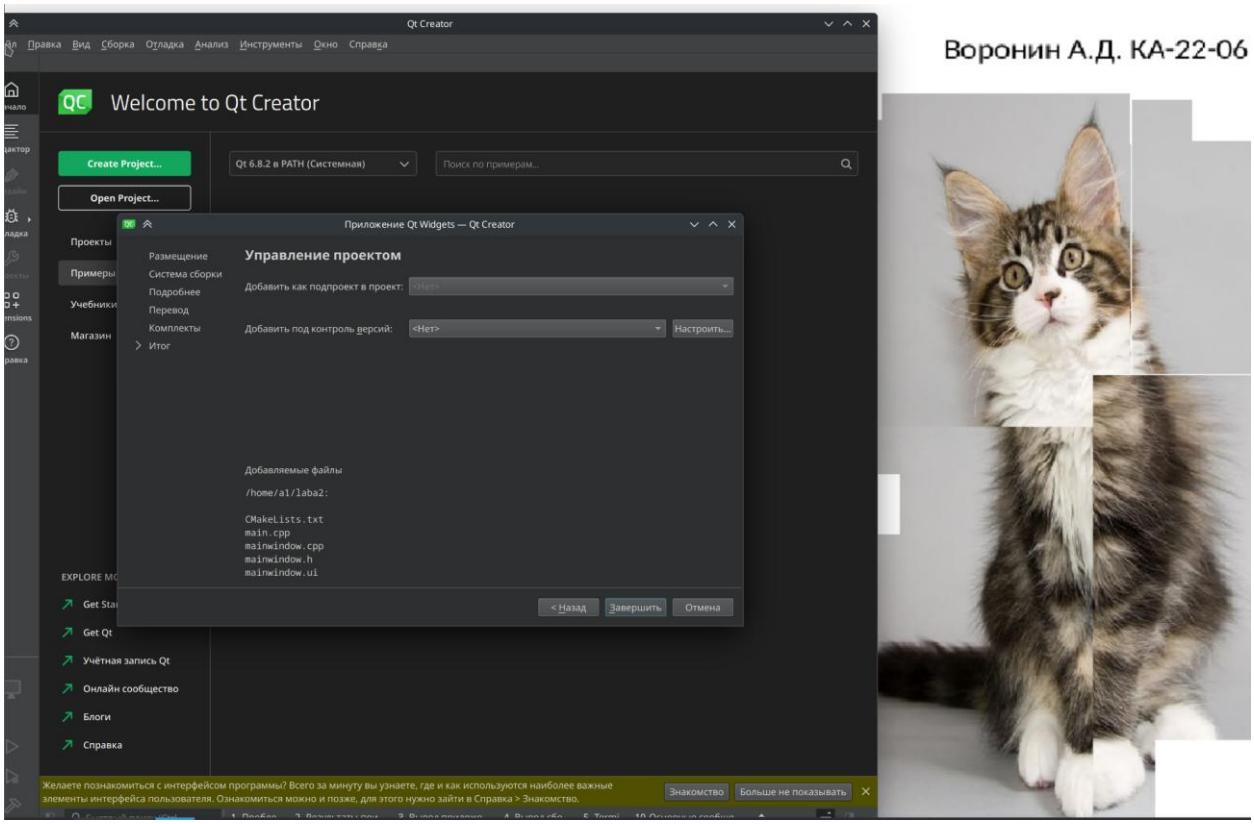

Рисунок 7 – Создание приложения

Добавьте на форму QLineEdit для ввода текста, QPushButton для обработки текста, QLabel для вывода результата.

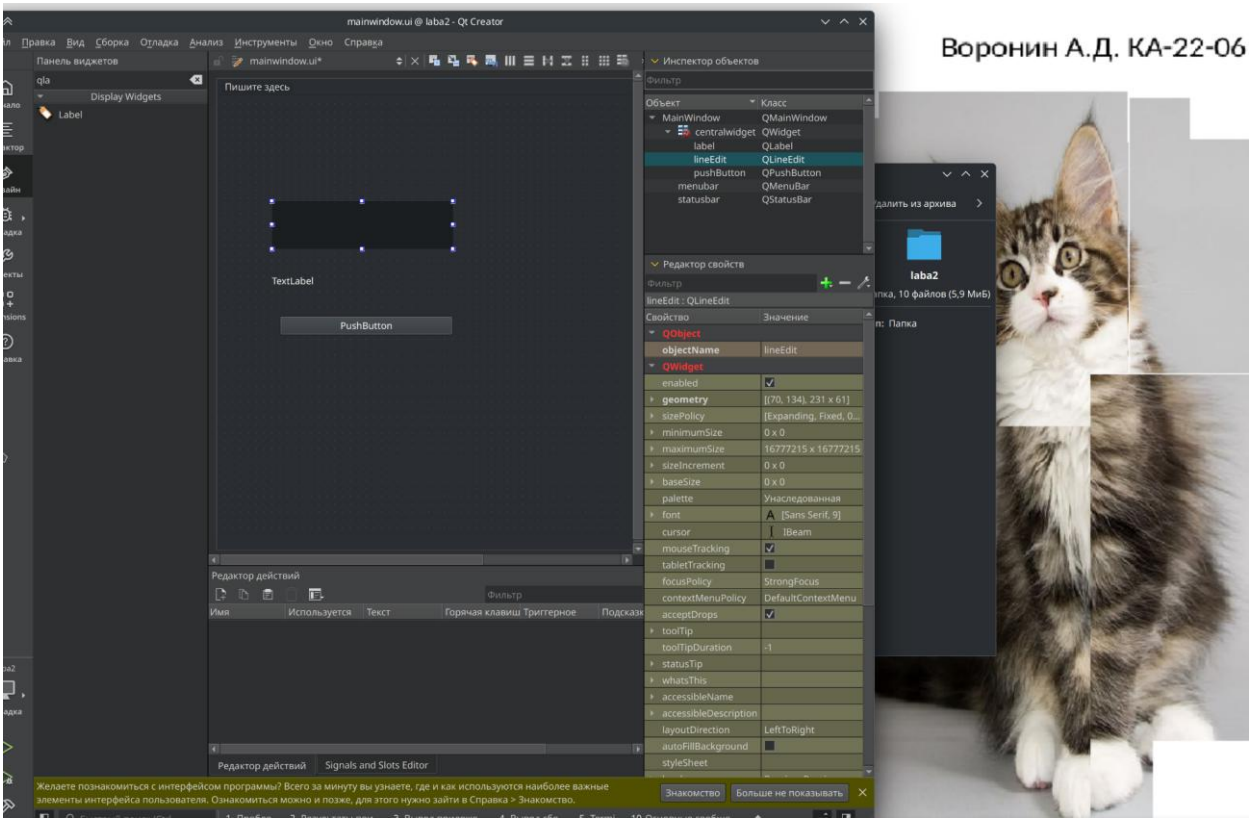

Рисунок 8 – Размещение виджетов

Шаг 2. Настройка объектов.

Выберите QLineEdit и в редакторе свойств найдите objectName в котором измините ему имя на inputLineEdit. Аналогично настройте имена для QPushButton и QLabel: processButton и resultLabel.

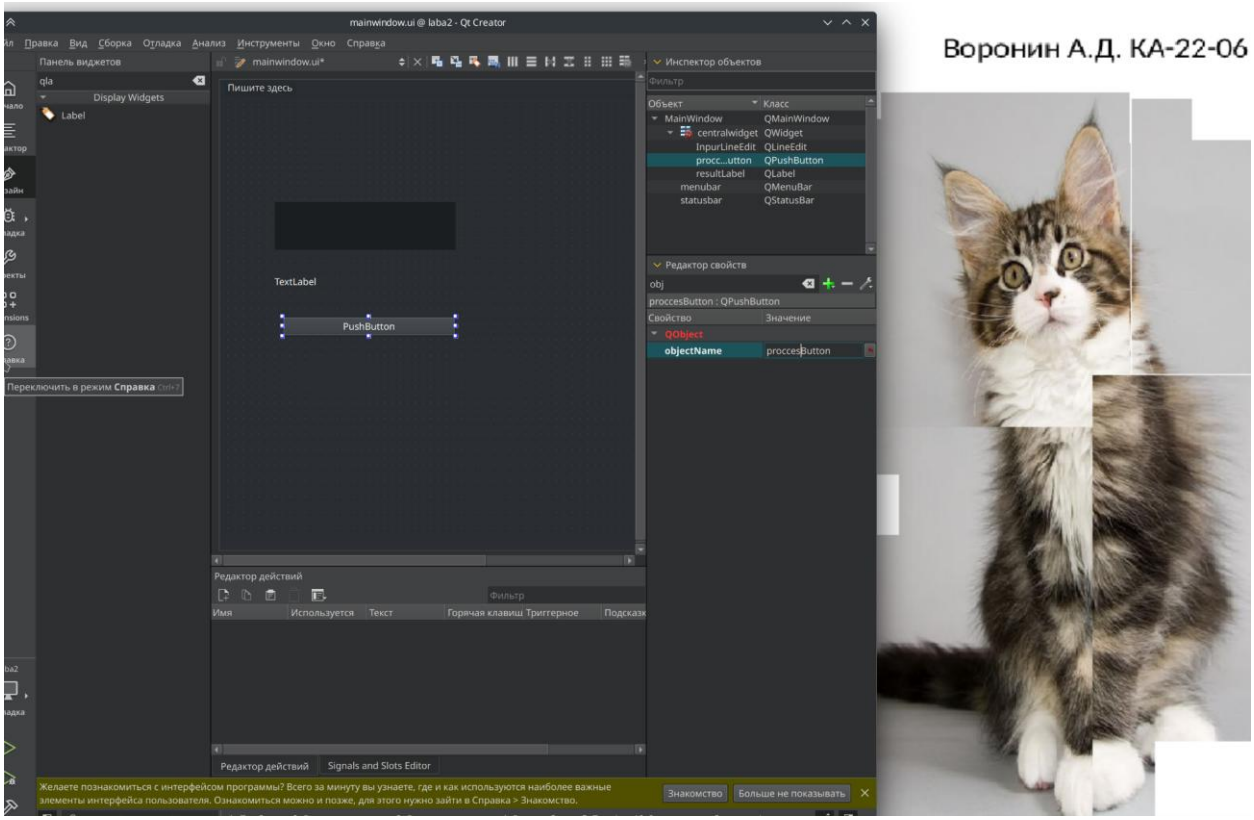

Рисунок 9 – Переименование виджетов

Шаг 3. Реализация слота для обработки.

Добавьте в mainwindow.h объявление вашего слота в секцию private slots или public slots (зависит от ваших требований к защите).

```
private:
    void processText();
```

Далее перейдите в mainwindow.cpp и реализуйте этот слот, где будет производиться обработка текста из QLineEdit, например, преобразование в верхний регистр.

```
void MainWindow::processText() {
    QString inputText = ui->inputLineEdit->text();
```

```
    QString processedText = inputText.toUpper();
    ui->resultLabel->setText(processedText);
}
```

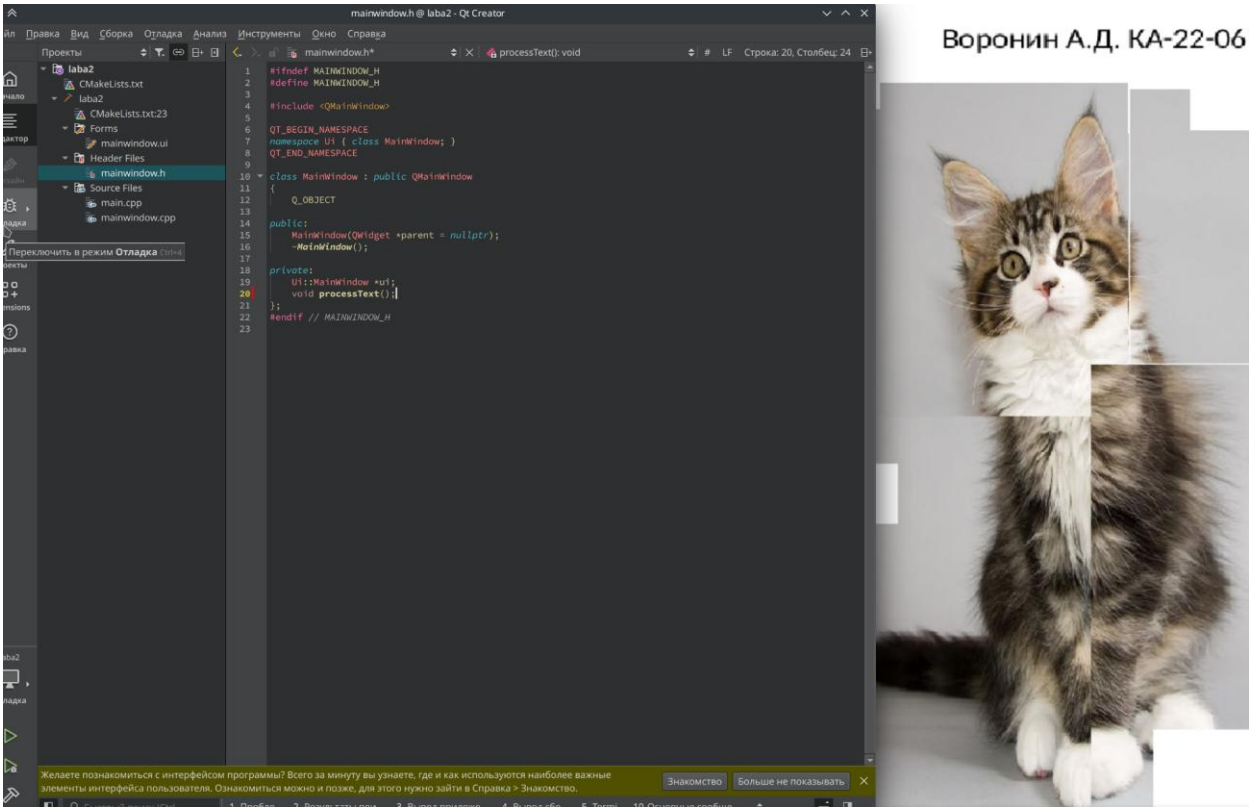

Рисунок 10 – Объявление слота

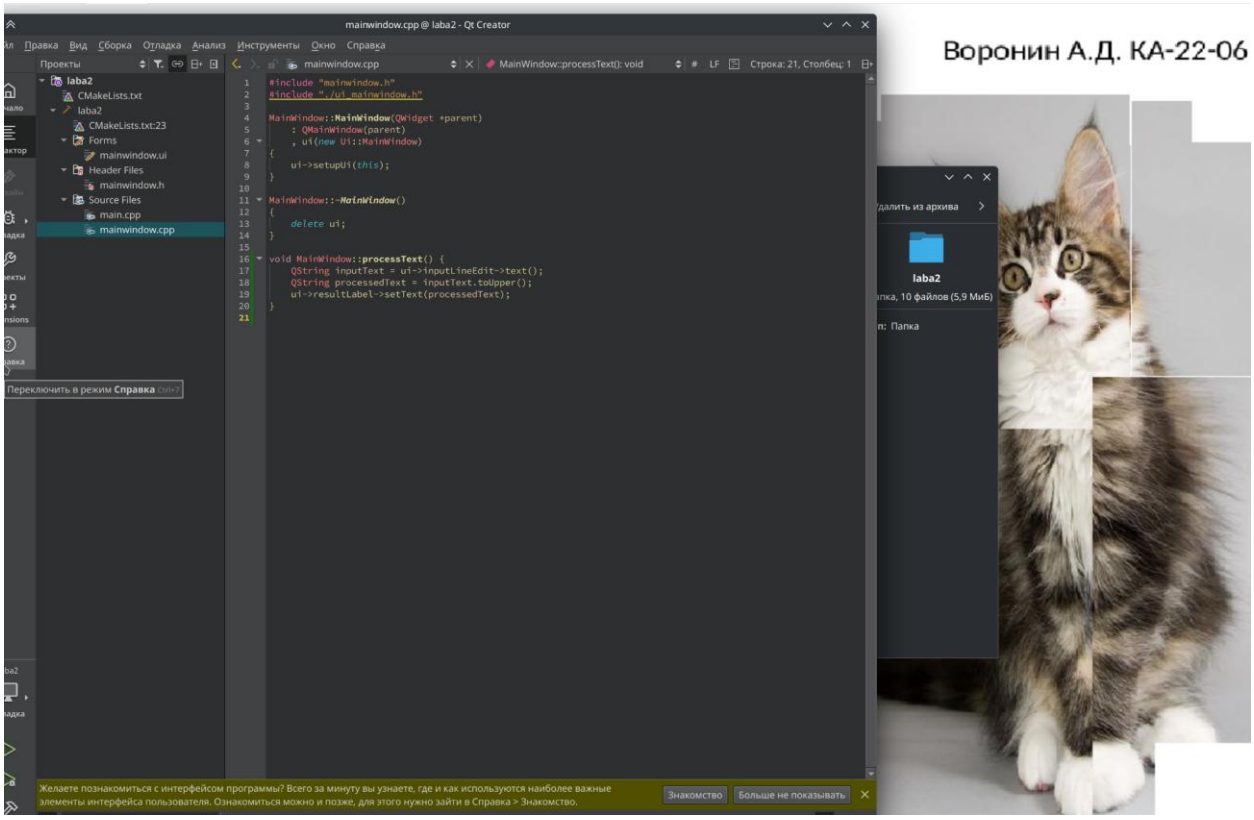

Рисунок 11 – Реализация слота

Шаг 4. Соединение сигналов и слотов.

В конструкторе MainWindow, соедините сигнал нажатия кнопки с вашим слотом с помощью следующей команды:

connect(ui->processButton, &QPushButton::clicked, this, &MainWindow::processText);

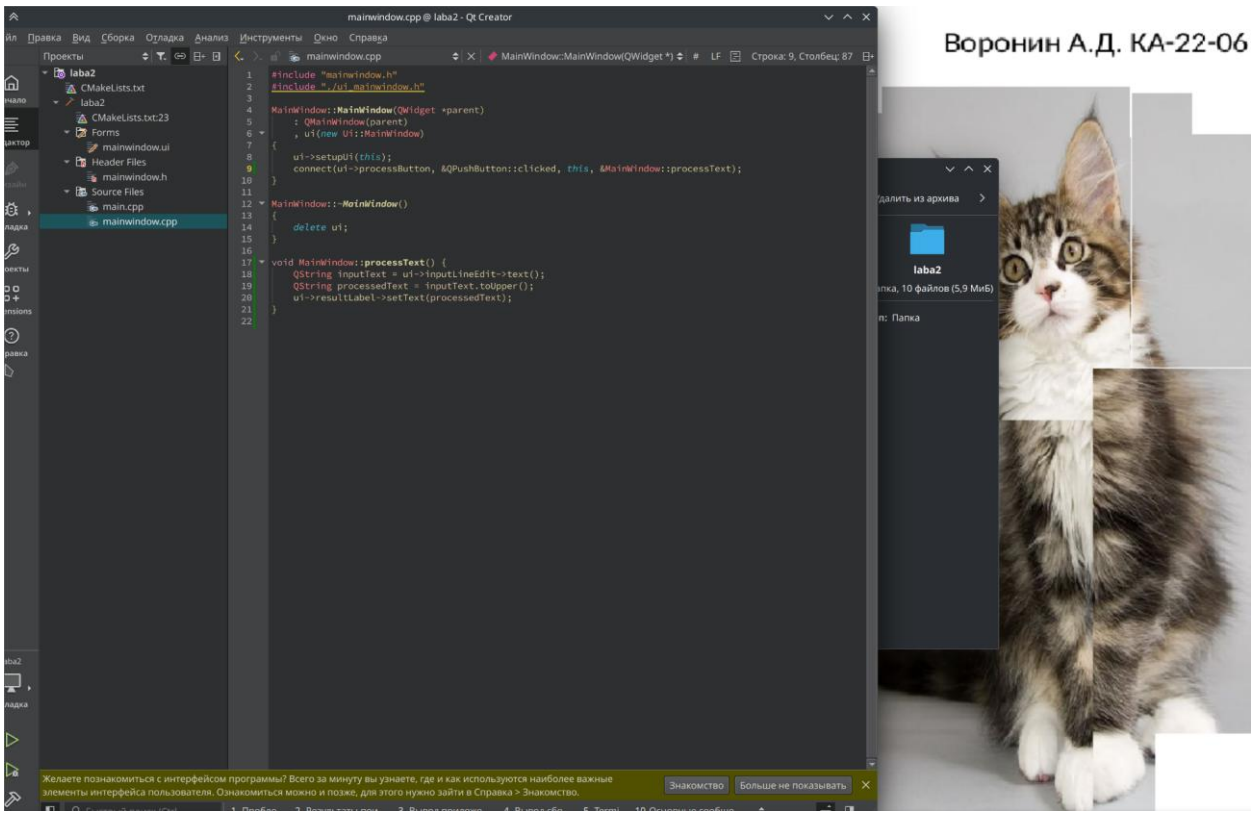

Рисунок 12 – Соединение сигнала

Шаг 5. Тестирование.

Соберем и запустим приложение, после чего проверим его обработку.

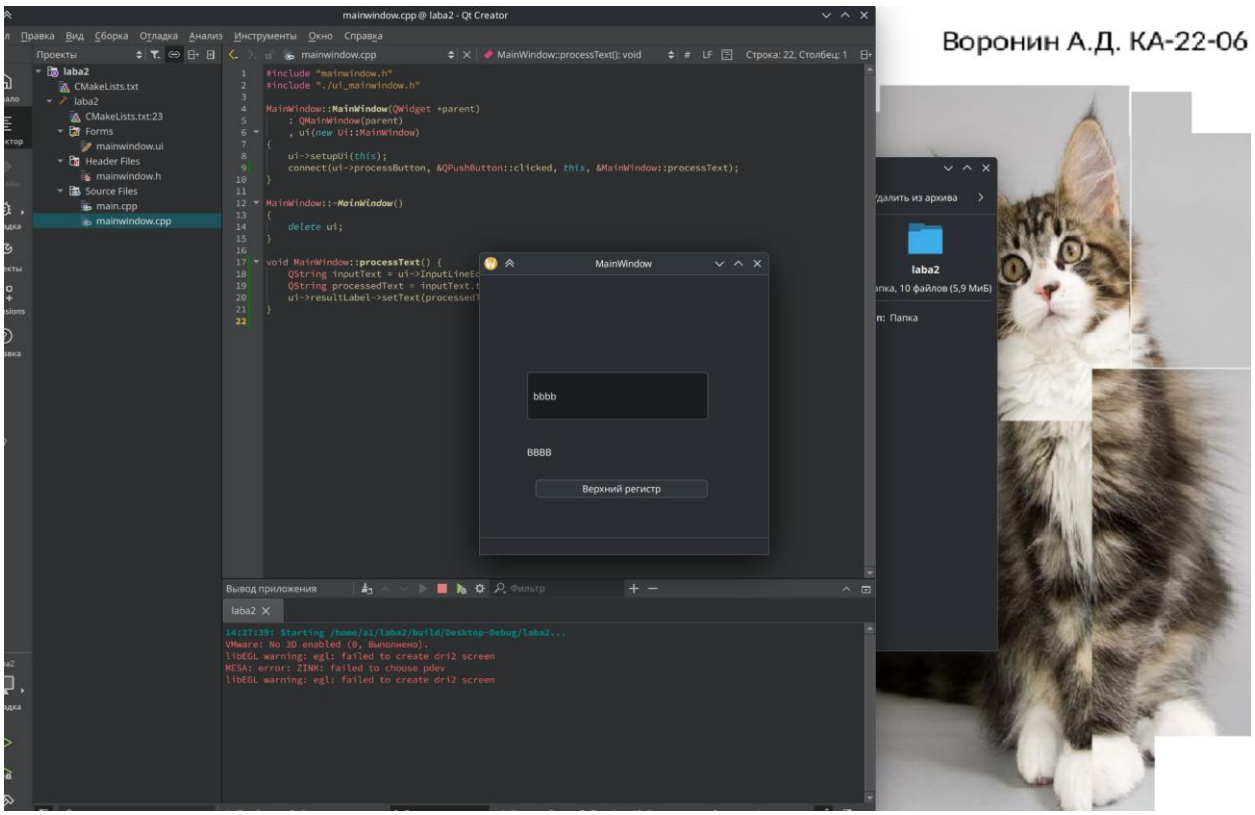

Рисунок 13 – Тестирование

ЧАСТЬ 2. Использование QVariant для работы с различными типами данных

Шаг 1. Реализация обработки QVariant.

Добавьте в mainwindow.h объявление вашего слота в секцию private slots или public slots (зависит от ваших требований к защите).

private:

void processVariant();

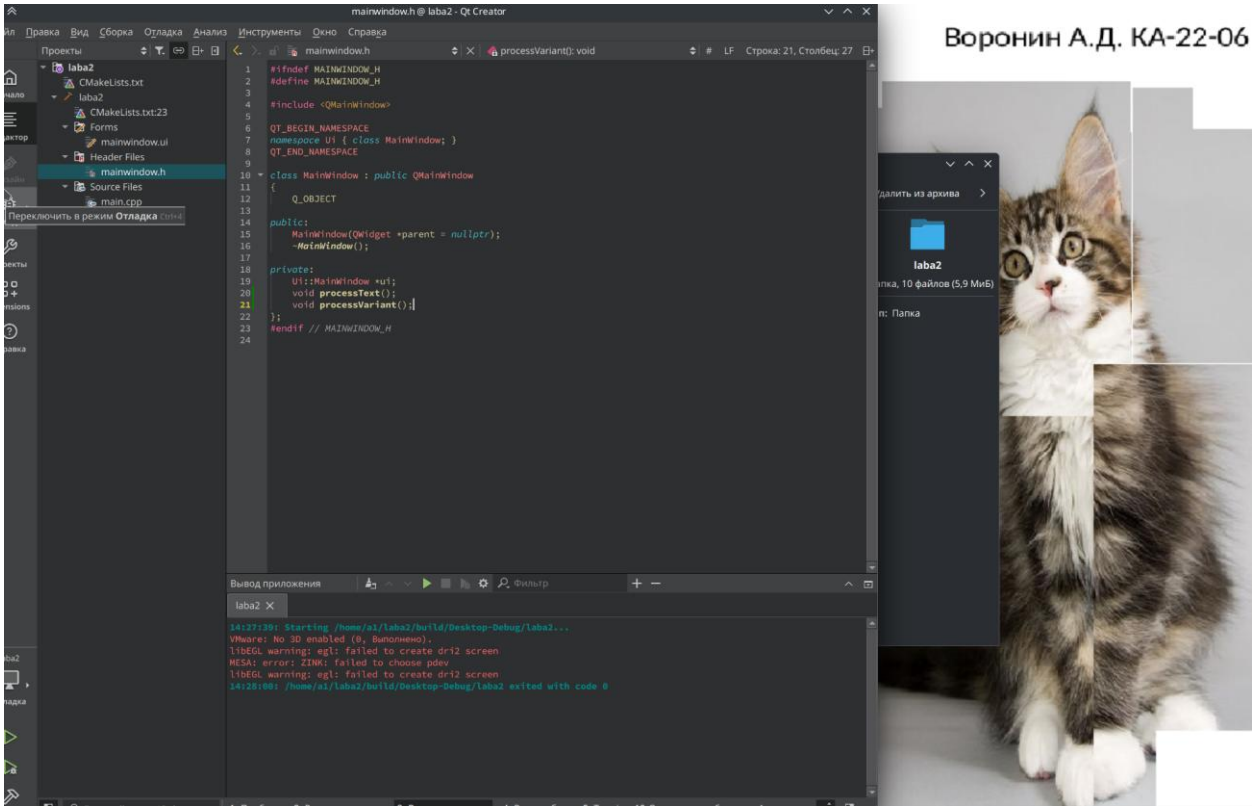

Рисунок 14 – Объявление слота

В конструкторе MainWindow, соедините сигнал нажатия кнопки с вашим слотом с помощью следующей команды:

connect(ui->processButton, &QPushButton::clicked, this, &MainWindow::processVariant);

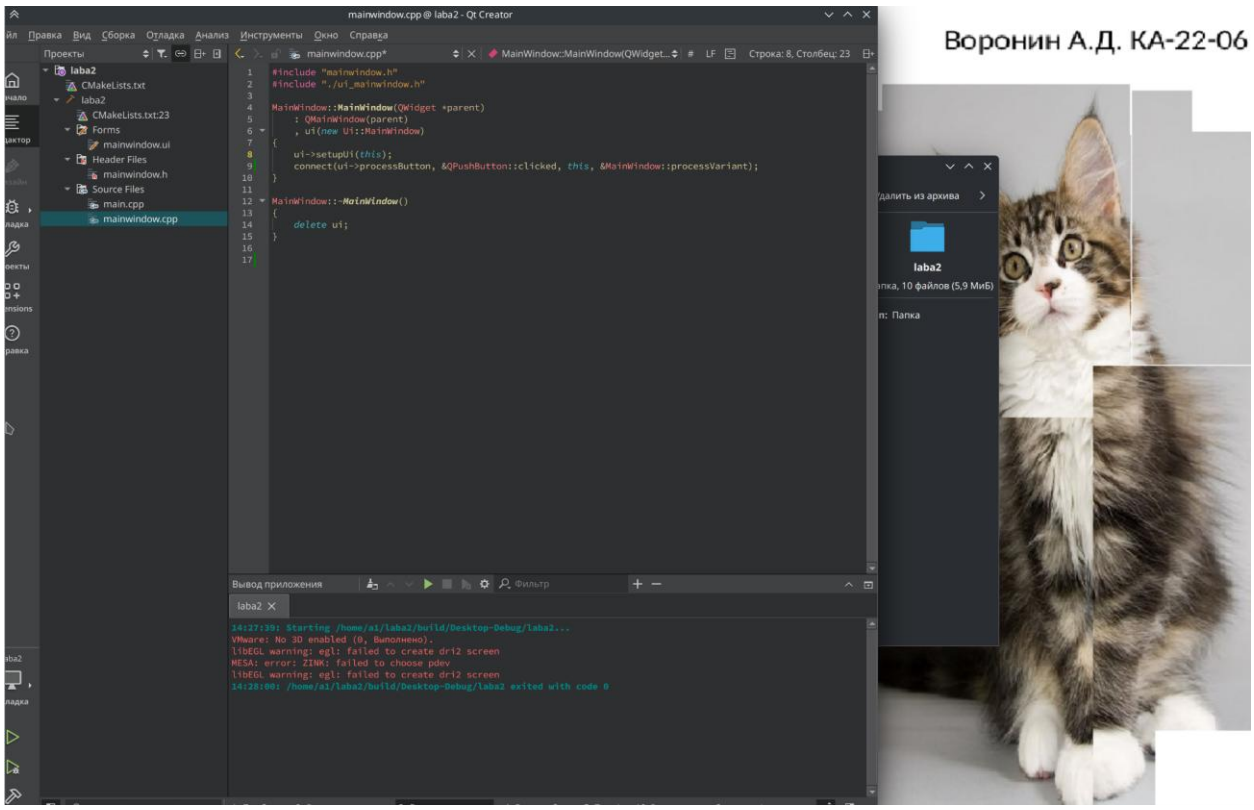

Рисунок 15 – Соединение сигнала

Далее перейдите в mainwindow.cpp и реализуйте этот слот, где будет производиться функция по условию задания.

```
#include <QDate>
#include <QDebug>

void MainWindow::processVariant() {
    QString inputText = ui->inputLineEdit->text();
    QVariant var;
    bool isNumber;
    int intValue = inputText.toInt(&isNumber);

    QDate dateValue = QDate::fromString(inputText, "dd.MM.yyyy");
    if (!dateValue.isValid()) {
        dateValue = QDate::fromString(inputText, "dd-MM-yyyy");
    }
```

```
    if (!dateValue.isValid()) {
        dateValue = QDate::fromString(inputText, "dd/MM/yyyy");
    }

    if (isNumber) {
        var = intValue;
    } else if (dateValue.isValid()) {
        var = dateValue;
    } else {
        var = inputText;
    }

    QString resultText;
    if (var.type() == QVariant::Int) {
        resultText = "Число: " + QString::number(var.toInt() * 2);
    } else if (var.type() == QVariant::Date) {
        resultText = "Дата: " + var.toDate().toString("dd.MM.yyyy");
    } else {
        resultText = "Строка: " + var.toString();
    }

    ui->resultLabel->setText(resultText);
}
```

Шаг 2. Тестирование.

Протестируем приложение, введя число, строку и дату.

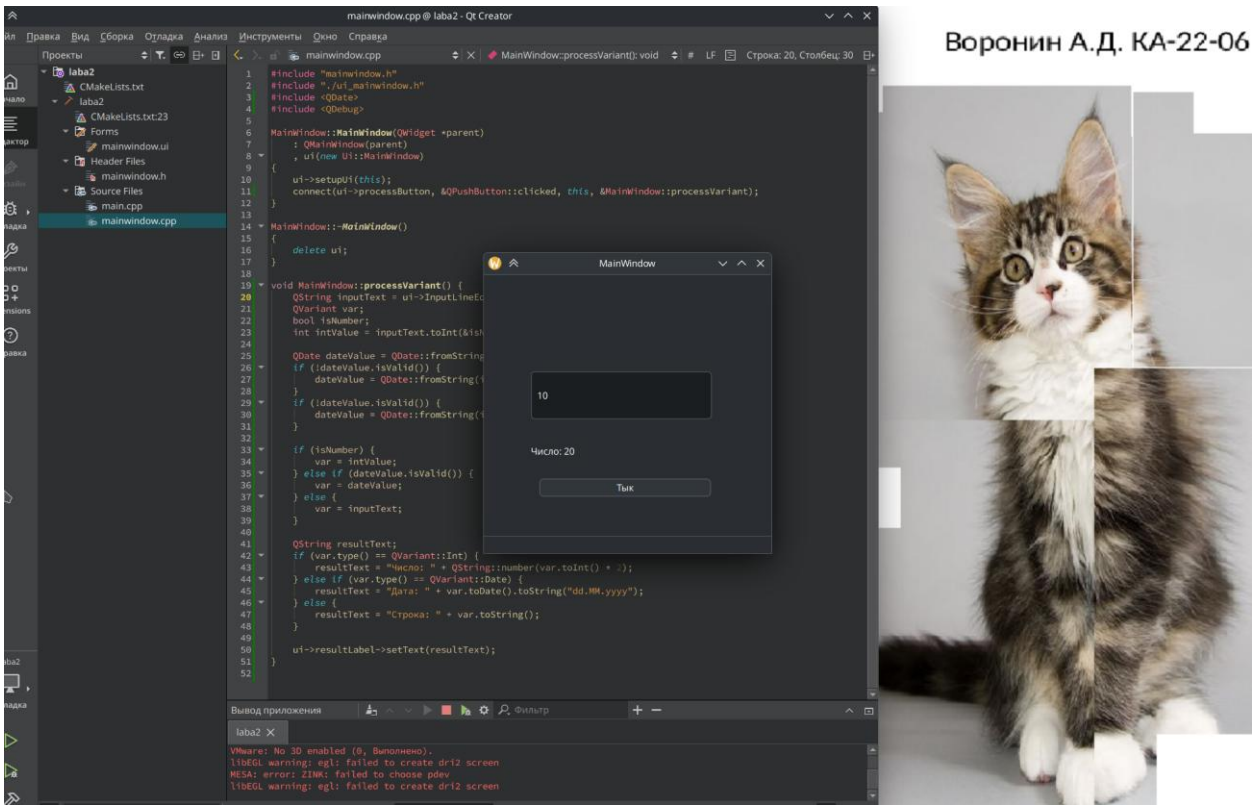

Рисунок 16 – Тест на число

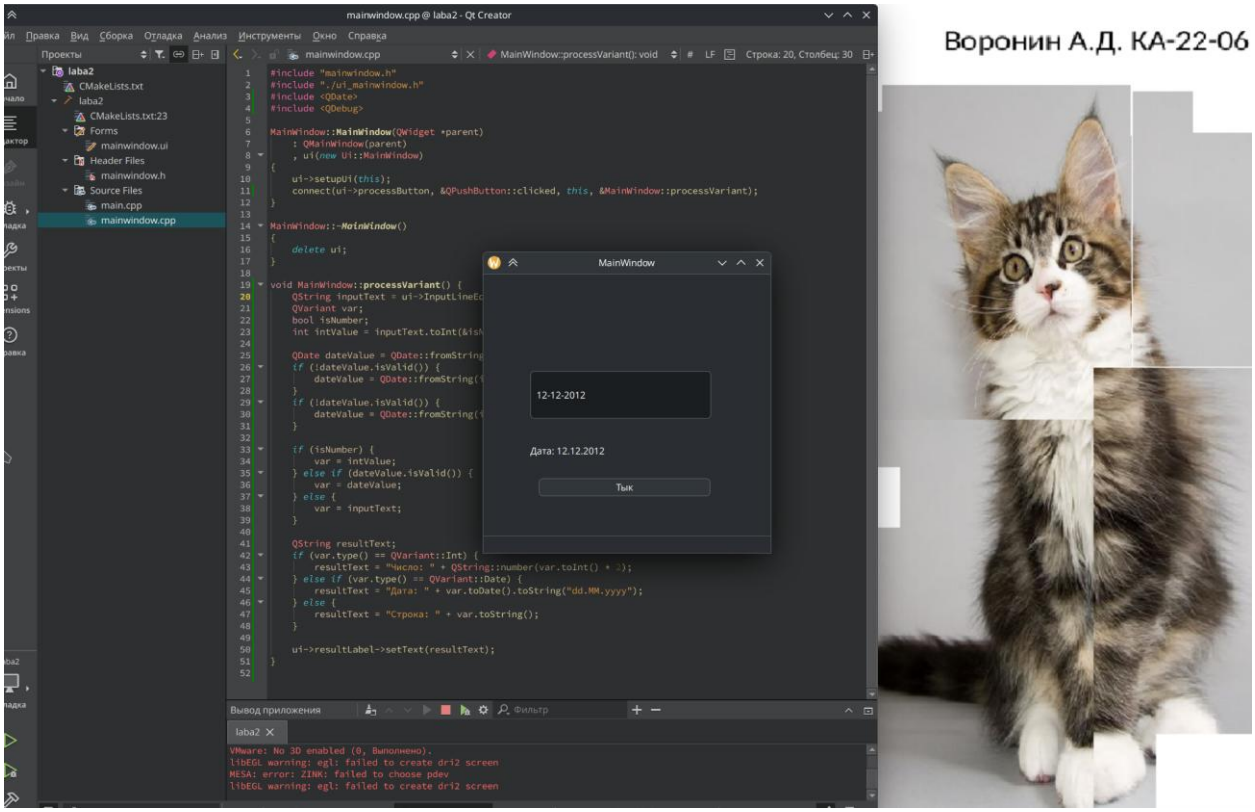

Рисунок 17 – Тест на дату

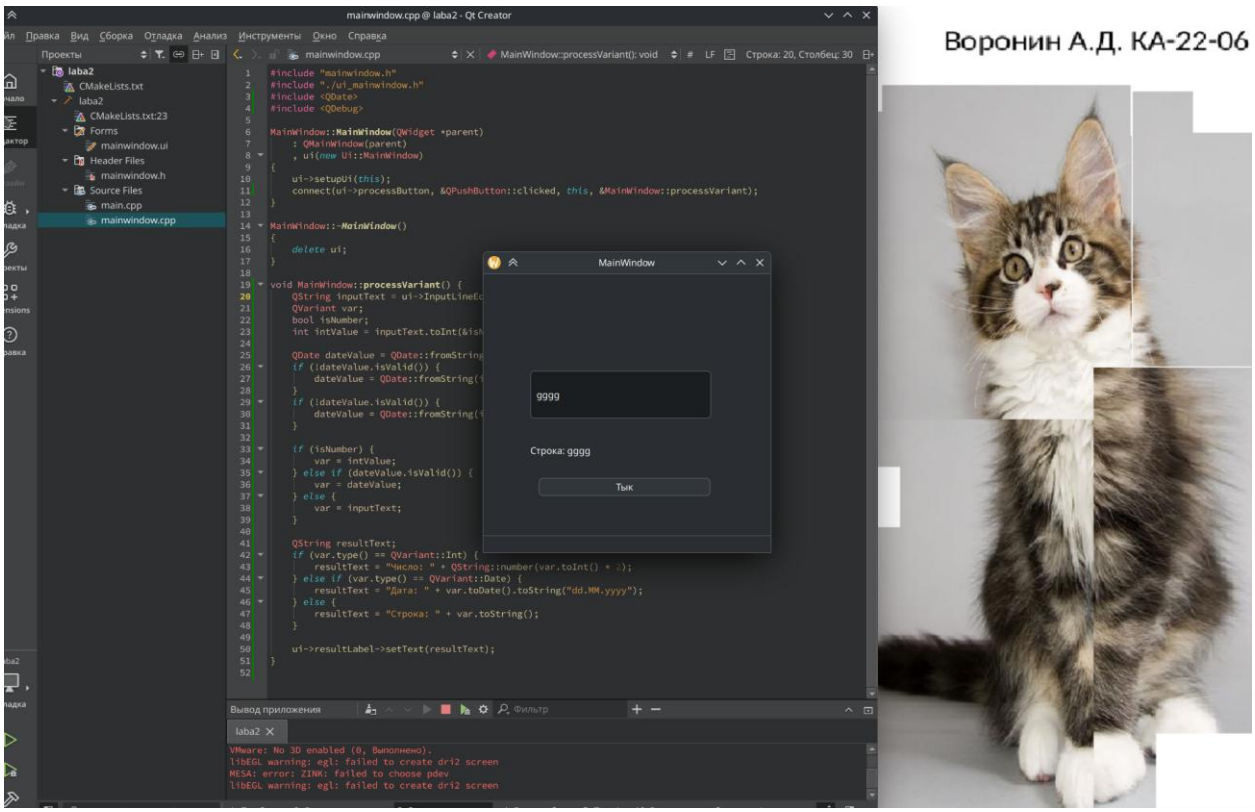

Рисунок 18 – Тест на строку

ЧАСТЬ 3. Создание собственных сигналов и слотов

Шаг 1. Создание нового класса

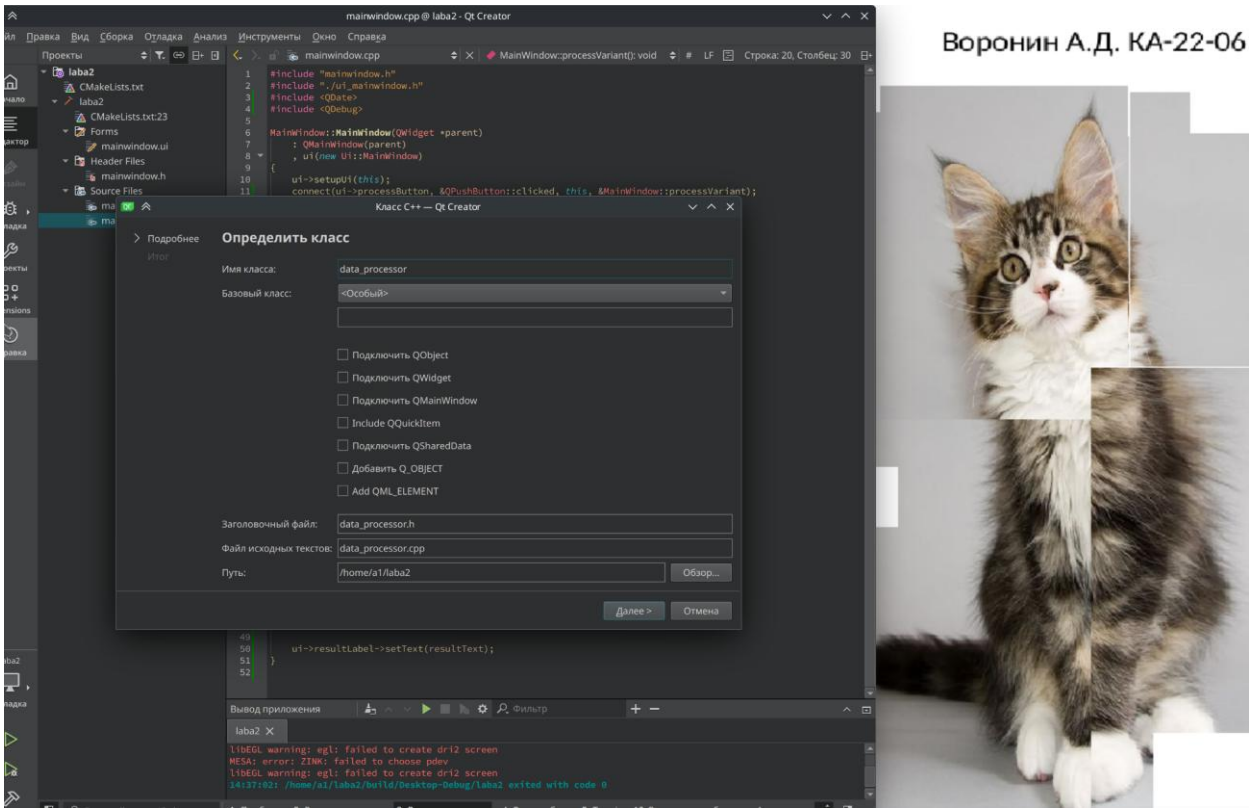

Рисунок 19 – Создание класса

Измените код в файле data_processor.h. Наследуйте его от QObject и добавьте сигнал с обработанными данными.

```
#ifndef DATA_PROCESSOR_H
#define DATA_PROCESSOR_H

#include <QObject>

class DataProcessor : public QObject {
    Q_OBJECT
public:
    explicit DataProcessor(QObject *parent = nullptr) : QObject(parent) {}
signals:
    void dataProcessed(const QString &result);
};

#endif // DATA_PROCESSOR_H
```

Шаг 2. Добавление метода для обработки данных

Добавьте метод, который выполняет обработку данных.

```
public slots:
    void processData(const QString &input);
```

В файле data_processor.cpp добавьте обработку данных.

```
#include "data_processor.h"

void DataProcessor::processData(const QString &input) {
    QString processedData = "Обработано: " + input;
    emit dataProcessed(processedData);
```

}

Зайдите в класс mainwindow.h, который отвечает за интерфейс и обработку событий и добавьте основные объекты, такие как handleProcessedData(слот для вывода обработанных данных), onProcessButtonClicked(слот для обработки нажатия кнопки), QLineEdit(поле ввода), QLabel(поля вывода), QPushButton(кнопка), DataProcessor(экземпляр обработчика данных).

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include "data_processor.h"

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
private slots:
    void handleProcessedData(const QString &data);
    void onProcessButtonClicked();
private:
    QLineEdit *inputField;
    QLabel *outputLabel;
    QPushButton *processButton;
    DataProcessor *processor;
};

#endif // MAINWINDOW_H
```

Шаг 3. Реализация логика вызова метода обработки данных

#include "mainwindow.h"

#include <QVBoxLayout>

```
#include <QWidget>

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    QWidget *centralWidget = new QWidget(this);
    setCentralWidget(centralWidget);

    inputField = new QLineEdit(this);
    processButton = new QPushButton("Обработать", this);
    outputLabel = new QLabel("Результат: ", this);

    QVBoxLayout *layout = new QVBoxLayout(centralWidget);
    layout->addWidget(inputField);
    layout->addWidget(processButton);
    layout->addWidget(outputLabel);

    processor = new DataProcessor(this);

    connect(processor, &DataProcessor::dataProcessed, this,
&MainWindow::handleProcessedData);
    connect(processButton, &QPushButton::clicked, this,
&MainWindow::onProcessButtonClicked);
}
void MainWindow::handleProcessedData(const QString &data) {
    outputLabel->setText("Результат: " + data);
}
void MainWindow::onProcessButtonClicked() {
    QString inputData = inputField->text();
```

```
    processor->processData(inputData);
}
```

Самостоятельная работа

В первом дополнительном задании необходимо расширить функциональность обработки текста, а именно:

- Добавить в приложение возможность выбирать несколько режимов обработки введённого текста (например, преобразование в нижний регистр, обратный порядок символов, удаление пробелов).
- Реализовать переключатель (QComboBox или набор радиокнопок) для выбора нужного режима обработки.
- Организовать сигналы и слоты так, чтобы при изменении режима обработки происходило обновление результата в реальном времени.

Для выполнения были внесены следующие изменения:

В mainwindow.h:

1. Был добавлен выпадающий список (QComboBox), необходимый для выбора режима обработки данных.

2. Добавлены новые слоты void processText(); – новый слот, который, выполняет обработку текста и void updateProcessingMode(int index); – новый слот, который изменяет режим обработки в зависимости от выбора в QComboBox.

В mainwindow.cpp:

1. Добавлен выпадающий список QComboBox, который позволяет пользователю выбирать режим обработки текста.

. Добавлены режимы обработки текста в QComboBox

. Добавлен новый слот processText()

4. Изменено подключение сигналов и слотов

```
 connect(modeSelector,
QOverload::of(&QComboBox::currentIndexChanged), this,
```

&MainWindow::updateProcessingMode);

Это обеспечивает автоматическую обработку текста при смене режима.

Во втором дополнительном задании нужно реализовать улучшение работы с QVariant:

• Расширить обработку ввода, добавив проверку не только для чисел и дат, но и для вещественных чисел и логических значений.

• Реализовать функцию, которая принимает QVariant, определяет его тип и выполняет соответствующее преобразование.

• Выводить результат обработки в QLabel с указанием типа данных.

На данном этапе в код внесены следующие изменения:

В mainwindow.h:

1. Добавлены методы для работы с QVariant: parseInput(const QString &input): определяет тип входных данных. formatVariant(const QVariant &var): форматирует QVariant в строку. processVariant(): вызывает обработку QVariant.

2. Обновлен QComboBox – добавлен новый режим "Определить тип (QVariant)".

В mainwindow.cpp:

1. Обновлена логика обработки данных: В onProcessButtonClicked() добавлена проверка режима: если выбран QVariant, вызывается processVariant().

2. Добавлена функция parseInput():, которая определяет целые, вещественные числа, булевы значения и даты.

3. Добавлена функция formatVariant():, которая обрабатывает QVariant в зависимости от типа входных данных.

Далее необходимо создать цепочку обработки при использовании собственных сигналов и слотов и разработать диалоговое окно для ввода и обработки данных.

Для реализации было внесено и добавлено следующее:

1. Добавлен новый класс AdvancedDataProcessor

Теперь обработка текста выполняется в несколько этапов:

- Обработка текста (processText) — приведение к нижнему
- регистру, обратный порядок строки или удаление пробелов.
- Форматирование (formatText) — добавление префикса "Отформатировано:".
- Логирование (logText) — вывод результата в консоль.

Передача данных организована через сигналы textProcessed, textFormatted, textLogged. После обработки результат передаётся в QLabel в MainWindow.

2. В Mainwindow организована цепочка сигналов и слотов

```
connect(advancedProcessor,
&AdvancedDataProcessor::textProcessed,
advancedProcessor, &AdvancedDataProcessor::formatText);
connect(advancedProcessor,
&AdvancedDataProcessor::textFormatted,
advancedProcessor, &AdvancedDataProcessor::logText);
connect(advancedProcessor,
&AdvancedDataProcessor::textLogged, this, this {
outputLabel->setText("Цепочка обработки: " + text);
});
```

Теперь после нажатия "Обработать" текст проходит через три этапа, и результат появляется в QLabel.

3. Добавлен класс InputDialog (в mainwindow.h и mainwindow.cpp)

- Теперь можно вводить текст и выбирать режим обработки в
- отдельном окне.
- После обработки данные отправляются в MainWindow.

- Добавлена кнопка Открыть диалог в MainWindow.

После закрытия диалогового окна обработанный текст отображается в QLabel с надписью «Результат из диалога:».

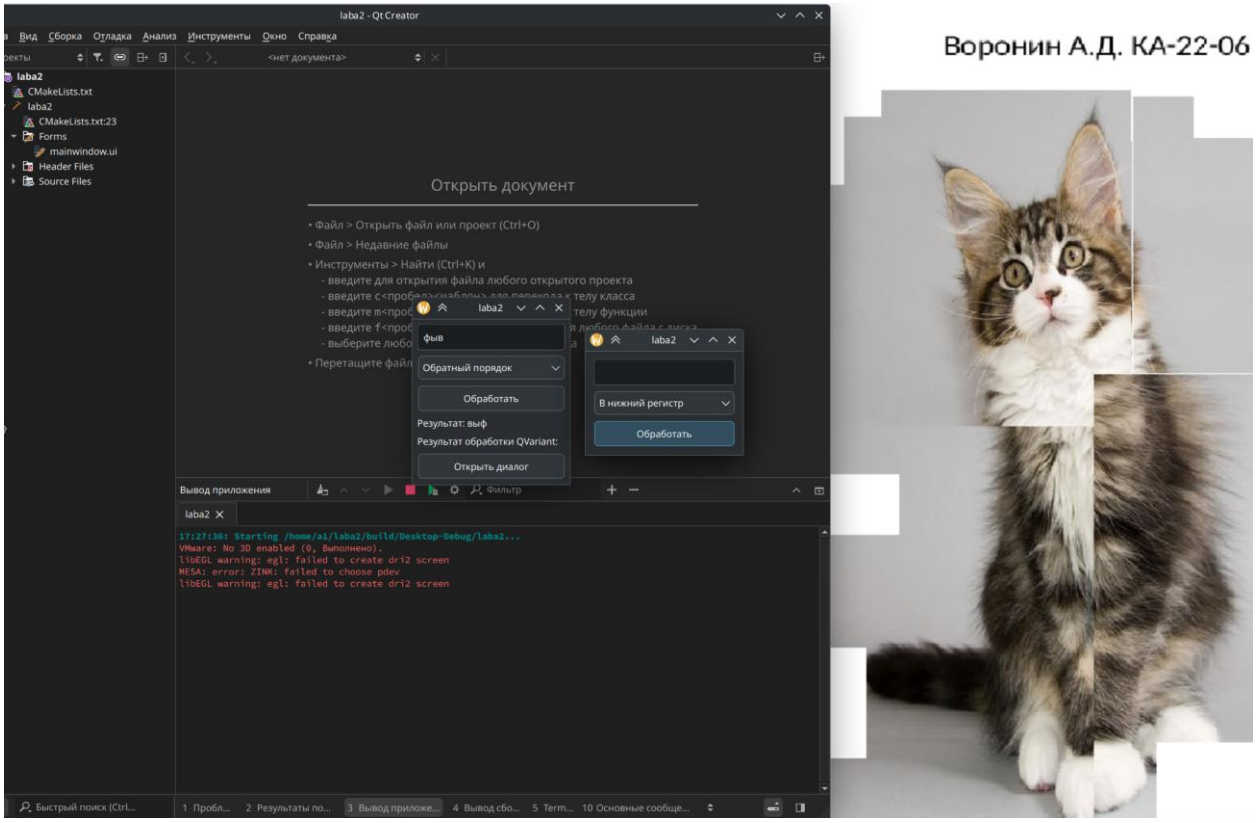

Рисунок 20 – Итоговое приложение

ЗАКЛЮЧЕНИЕ

В заключение, подводя итог проделанной работе, все поставленные в начале цели и задания данной лабораторной работы были достигнуты в полном объеме.

1. Что представляют собой сигналы и слоты в Qt?

Сигналы и слоты в Qt — это механизм обратного вызова, который позволяет объектам взаимодействовать между собой.

Сигналы — это события, которые объекты могут отправлять. Они не требуют указания получателя.

Слоты — это функции, которые обрабатывают сигналы.

Когда сигнал испускается, все подключенные к нему слоты автоматически вызываются. Это позволяет легко организовывать связь между компонентами без жесткой привязки объектов друг к другу.

2. Как используется класс QVariant и в каких случаях он может быть полезен?

QVariant — это универсальный контейнер для хранения данных разных типов. Он особенно полезен, когда заранее неизвестен тип данных, с которым предстоит работать.

QVariant может использоваться в следующих ситуациях:

1) Хранение и передача данных разных типов.

2) Работа с моделями (В QAbstractItemModel, где ячейки таблиц или списков могут содержать данные разных типов).

3) Использование в сигналах и слотах (Иногда слоты могут принимать данные разного типа, и использование QVariant упрощает обработку).

4) Сохранение настроек приложения (QSettings использует QVariant для хранения конфигурационных параметров, поскольку их тип может различаться).

3. Какие существуют способы подключения сигналов к слотам и в чем преимущества нового синтаксиса подключения (с использованием указателей на методы)?

В Qt существуют два основных способа подключения сигналов к слотам:

1) Старый (строковый) синтаксис (Qt 4 и ранее)

Использует макрос SIGNAL() и SLOT(), передавая их как строки.

connect(sender, SIGNAL(signalName(QString)), receiver, SLOT(slotName(QString)));

2) Новый (современный) синтаксис (Qt 5 и новее)

Использует указатели на функции (методы).

connect(sender, &SenderClass::signalName, receiver, &ReceiverClass::slotName);

Преимущества:

- Проверка типов во время компиляции – ошибки в сигналах/слотах обнаруживаются сразу.
- Более высокая производительность – нет затрат на строковую обработку.

4. Каковы основные принципы создания собственных сигналов и слотов в классах, наследуемых от QObject?

1. Наследование от QObject и использование макроса Q_OBJECT

Каждый класс, который использует сигнально-слотовый механизм, должен быть унаследован от QObject и содержать макрос Q_OBJECT (если есть сигналы или слоты).

2. Определение сигналов (signals). Сигналы объявляются в секции signals, но не имеют реализации — Qt сам обрабатывает их во время выполнения.

3. Определение слотов (slots) Слоты объявляются в секции public slots, protected slots или private slots. Они — обычные методы класса, которые можно вызывать как напрямую, так и через механизм connect().

4. Подключение сигналов к слотам (connect()) Подключение выполняется через QObject::connect().