

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра	<u>О7</u>	<u>Информационные системы и программная инженерия</u>
	шифр	наименование кафедры, по которой выполняется работа
Дисциплина	<u>Визуальное программирование</u>	<u></u>
		наименование дисциплины

ПРАКТИЧЕСКАЯ РАБОТА № 2

Создание собственных элементов управления

и работа с ними

Язык: C++ Qt

Вариант №10

ОБУЧАЮЩИЙСЯ

группы О726Б

Махов Н.М.

подпись

фамилия и инициалы

дата сдачи

ПРОВЕРИЛ

Преподаватель

ученая степень, ученое звание, должность

Устиновский Г.С.

подпись

фамилия и инициалы

Оценка / балльная оценка

дата проверки

СОДЕРЖАНИЕ

1	Цель работы и постановка задачи	3
1.1	Цель работы	3
1.2	Постановка задачи	3
1.3	Вариативная часть задания	3
2	Реализация	5
2.1	Содержание файла mywidget.cpp	5
2.2	Содержание файла mywidget.h	8
2.3	Содержание файла mainwindow.h	10
2.4	Содержание файла mainwindow.cpp	11
2.5	Содержание файла main.cpp	11
3	Демонстрация работы программы	14

1 Цель работы и постановка задачи

1.1 Цель работы

Научиться создавать новые элементы пользовательского интерфейса, представленные в виде отдельных компонентов для использования их в различных проектах.

1.2 Постановка задачи

Требуется разработать собственный элемент управления, выполненный в виде динамической библиотеки (в случае использования Qt или Windows Forms) или пакета NuGet.

Далее необходимо написать небольшую программу, демонстрирующую использование данного элемента управления. Важно реализовать взаимодействие с созданным элементом управления из тестовой программы – элемент управления должен передавать в основную программу данные и принимать из неё команды либо в формате слотов и сигналов, либо через вызовы методов и подключение обработчика событий.

Созданный элемент управления должен быть подключен к тестовой программе. Должна быть продемонстрирована возможность работы с ним с использованием дизайнера для создания пользовательских интерфейсов.

Формулировки вариантов используют термины из фреймворка Qt. При выполнении работы средствами языка C#, требуется найти аналогичные элементы управления из числа стандартных для создания собственного композитного элемента управления. Вместо испускания сигнала потребуются добавить возможность назначения внешнего по отношению к создаваемому контролю обработчика события, который сможет получать указанное в варианте свойство элемента.

1.3 Вариативная часть задания

Вариант №10 – Виджет конструктор анкет

Виджет представляет из себя текстовое поле, подпись к нему и кнопку ввод. Нажатие на кнопку ввода проверяет о наличии значений в полях и при успехе, испускает сигнал о успехе. Необходимо организовать

метод, добавляющий новое текстовое поле и подпись к нему. А также метод, позволяющий получить информацию как из всех полей, так и из конкретного. Метод хранения полей оставлен на усмотрение разработчика.

2 Реализация

Виджет создан как библиотеке C++, сборка через qmake, виджет написан с использованием .ui файла. Далее виджет был подключен к Qt Application, внутри проекта была протестирована его работоспособность, для вывода данных из виджета использовал QTabletWidget и QTextEdit.

2.1 Содержание файла mywidget.cpp виджета

```
#include "mywidget.h"
#include <QHBoxLayout>
#include <QMessageBox>
#include "ui_mywidget.h"

MyWidget::MyWidget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::MyWidget)
{
    ui->setupUi(this);

    // кнопка добавления поля
    connect(ui->addFieldButton, &QPushButton::clicked, [=]()
    {
        addField(QString("Field %1").arg(fields.size() + 1));
    });

    // кнопка отправки к слоту обработки
    connect(ui->submitButton, &QPushButton::clicked, this,
    &MyWidget::onSubmitClicked);
}

MyWidget::~MyWidget()
{

```

```

        delete ui;
    }

    // метод для добавления нового текстового поля и кнопки
    // удаления
    void MyWidget::addField(const QString &labelText)
    {
        // контейнер для поля, подписи и кнопки удаления
        QWidget *container = new QWidget(this);
        QHBoxLayout *layout = new QHBoxLayout(container);

        QLabel *label = new QLabel(labelText, this);
        QLineEdit *lineEdit = new QLineEdit(this);
        QPushButton *removeButton = new QPushButton("Remove",
this);

        layout->addWidget(label);
        layout->addWidget(lineEdit);
        layout->addWidget(removeButton);

        ui->fieldsLayout->addWidget(container);

        fields[labelText] = lineEdit;
        fieldContainers[labelText] = container;

        // подключаем кнопку удаления к слоту removeField
        connect(removeButton, &QPushButton::clicked, [=]() {
removeField(labelText); });
    }

```

```

// считывание данных из всех полей
QMap<QString, QString> MyWidget::getAllFieldsData() const
{
    QMap<QString, QString> fieldData;
    for (auto it = fields.begin(); it != fields.end(); ++it)
    {
        fieldData[it.key()] = it.value()->text();
    }
    return fieldData;
}

QString MyWidget::getFieldData(const QString &labelText)
const
{
    if (fields.contains(labelText)) {
        return fields[labelText]->text();
    }
    return QString();
}

// удаление текстового поля
void MyWidget::removeField(const QString &labelText)
{
    if (fields.contains(labelText)) {
        QWidget *container = fieldContainers[labelText];
        delete container;

        fields.remove(labelText);
    }
}

```

```

        fieldContainers.remove(labelText);
    }
}

void MyWidget::onSubmitClicked()
{
    bool allFilled = true;
    for (auto it = fields.begin(); it != fields.end(); ++it)
    {
        if (it.value()->text().isEmpty()) {
            allFilled = false;
            break;
        }
    }

    if (allFilled) {
        QMap <QString, QString> formData = getAllFieldsData();
        emit formSubmittedWithData(formData);
        QMessageBox::information(this, "Success", "All fields
are filled!");
    } else {
        QMessageBox::warning(this, "Error", "Please fill all
fields!");
    }
}

```

2.2 Содержание файла mywidget.h виджета

```

#ifndef MYWIDGET_H
#define MYWIDGET_H
#include <QHBoxLayout>
#include <QLabel>

```



```

#include <QLineEdit>
#include <QMap>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWidget>
#include "myWidget_global.h"
QT_BEGIN_NAMESPACE
namespace Ui {
class MyWidget;
}
QT_END_NAMESPACE

class MYWIDGET_EXPORT MyWidget : public QWidget
//MYWIDGET_EXPORT
{
    Q_OBJECT

public:
    explicit MyWidget(QWidget *parent = nullptr);
    ~MyWidget();

    void addField(const QString &labelText);
    QMap<QString, QString> getAllFieldsData() const;
    QString getFieldData(const QString &labelText) const;

signals:
    // сигнал об успешной отправке формы
    void formSubmittedWithData(const QMap<QString, QString>
&data);

```

```

private slots:
    void onSubmitClicked();
    void removeField(const QString &labelText);

private:
    Ui::MyWidget *ui;
    QMap<QString, QLineEdit *> fields; // сохранение полей с
их названиями
    QMap<QString, QWidget *> fieldContainers; // для хранения
контейнеров полей
};
#endif // MYWIDGET_H

```

2.3 Содержание файла mainwindow.h приложения с собственным виджетом

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTextEdit>
#include <QTableWidget>
#include "mywidget.h"
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

```

```

private slots:
    //слот для обработки данных из MyWidget
    void displayFormData(const QMap<QString, QString> &data);
private:
    MyWidget *myWidget;
    QTextEdit *textEdit;
    QTableWidgetItem *tableWidget;
};
#endif // MAINWINDOW_H

```

2.4 Содержание файла mainwindow.cpp приложения с собственным виджетом

И так далее

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QSplitter>
#include <QVBoxLayout>
#include <QHeaderView>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{

    myWidget = new MyWidget(this);
    textEdit = new QTextEdit(this);
    textEdit->setReadOnly(true);

    tableWidget = new QTableWidgetItem(0, 2, this);
    tableWidget->setHorizontalHeaderLabels(QStringList() <<
"Field" << "Data");

```

```

        tableWidget->horizontalHeader()-
>setStretchLastSection(true);
        tableWidget->horizontalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);
        // используем QSplitter для разделения окна на две части
        QSplitter *splitter = new QSplitter(this);
        splitter->addWidget(myWidget);
        QWidget *container = new QWidget(this);
        QVBoxLayout *layout = new QVBoxLayout(container);
        layout->addWidget(textEdit);
// первый виджет - текстовое поле
        layout->addWidget(tableWidget);
// второй виджет - таблица
        splitter->addWidget(container);

        splitter->setStretchFactor(0, 1);
        splitter->setStretchFactor(1, 1);

        setCentralWidget(splitter);
        connect(myWidget, &MyWidget::formSubmittedWithData, this,
&MainWindow::displayFormData);
    }
    MainWindow::~MainWindow(){}
    void MainWindow::displayFormData(const QMap<QString, QString>
&data)
    {
        textEdit->clear();
        tableWidget->setRowCount(0);
        QString displayText;

```

```

        for (auto it = data.begin(); it != data.end(); ++it) {
            displayText += QString("%1:
%2\n").arg(it.key()).arg(it.value());
        }
        textEdit->setText(displayText);
        int row = 0;
        for (auto it = data.begin(); it != data.end(); ++it, ++row)
        {
            tableWidget->insertRow(row);
            tableWidget->setItem(row, 0, new
QTableWidgetItem(it.key()));
            tableWidget->setItem(row, 1, new
QTableWidgetItem(it.value()));
        }
    }
}

```

2.5 Содержание файла main.cpp приложения с собственным виджетом

```

#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Form Generator Application");
    w.show();
    return a.exec();
}

```

3 Демонстрация работы программы

Протестируем созданный виджет внутри проекта, для этого добавим `main.cpp` и создадим виджет. Также надо изменить в `.pro` `lib` на `app`, запускаем программы, видим, что виджет работает. Результат представлен на рисунке 1.

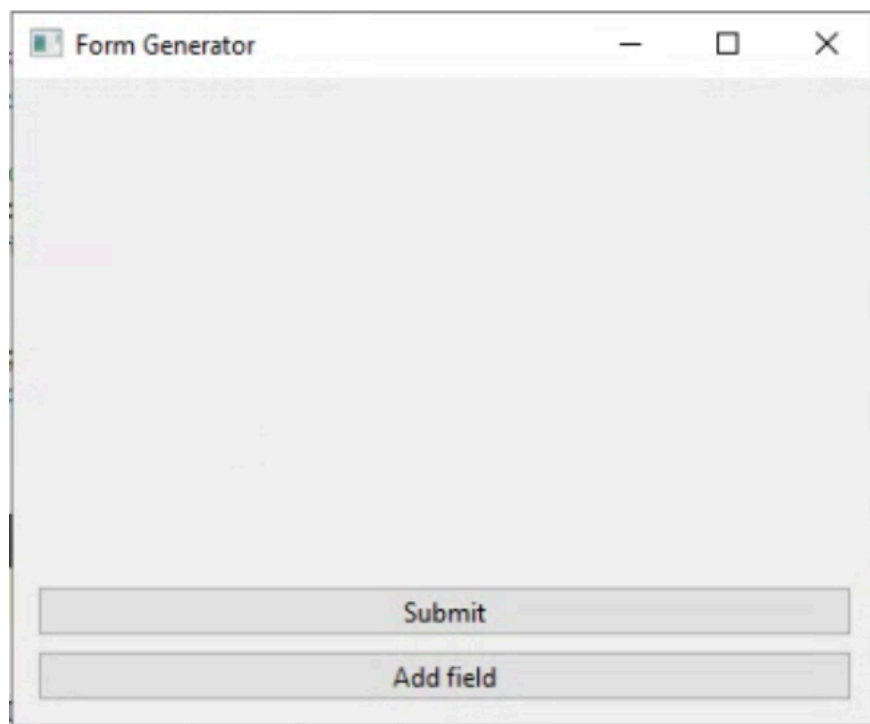


Рисунок 1 – Разработанный виджет

Добавим полей анкете. Результат представлен на рисунке 2.

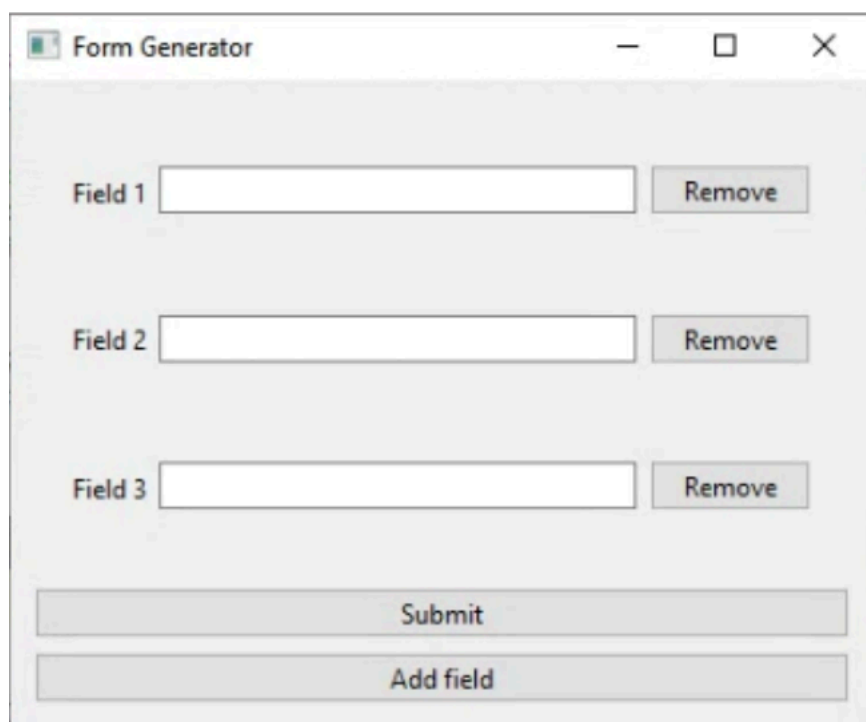
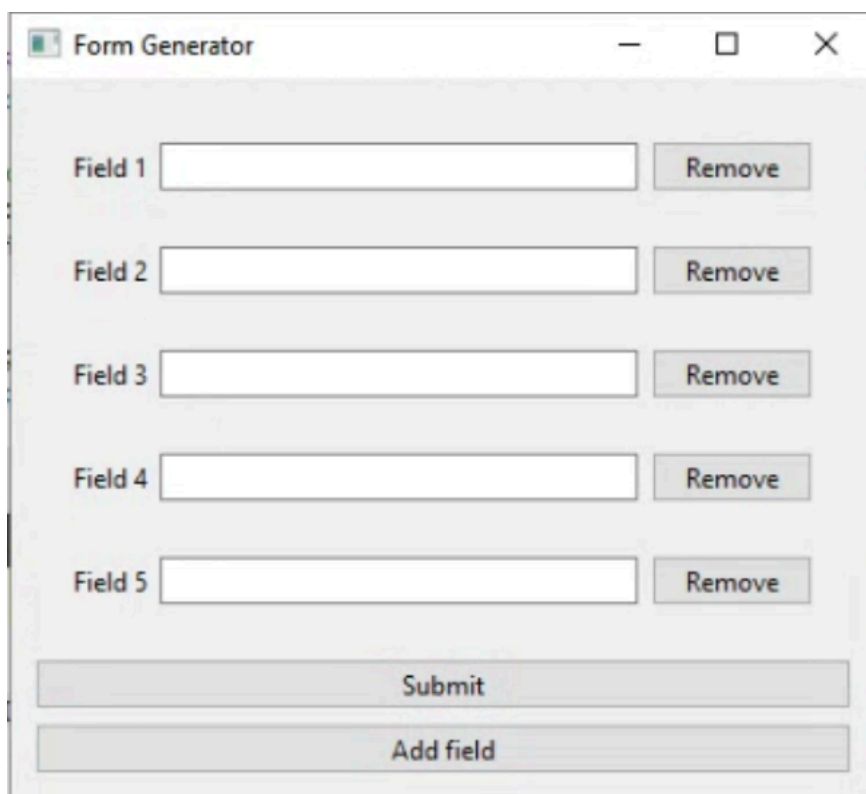


Рисунок 2 – Поля анкеты для ввода данных

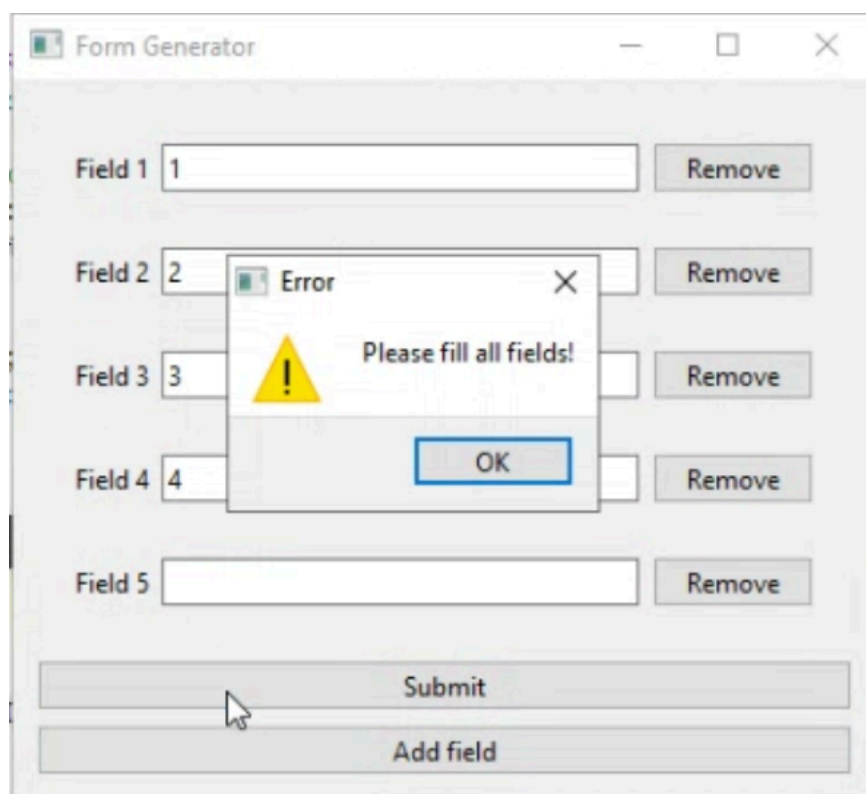
Окно динамически меняет размер при добавлении новых полей. Результат представлен на рисунке 3.



The screenshot shows a window titled "Form Generator". Inside, there are five rows, each containing a text input field and a "Remove" button to its right. The fields are labeled "Field 1" through "Field 5". At the bottom of the window, there are two buttons: "Submit" and "Add field".

Рисунок 3 – Динамическое изменение размеров окна по мере добавления полей анкеты

Проверка на наличие пустых полей, рисунок 4.



The screenshot shows the same "Form Generator" window, but now with an error dialog box open in the center. The dialog box has a yellow warning icon and the text "Please fill all fields!". The background form shows four fields filled with numbers 1 through 4, and one empty field. The "Submit" button is highlighted by a mouse cursor.

Рисунок 4 – Проверка на заполненность полей

Удалим Field 3. Результат представлен на рисунке 5.

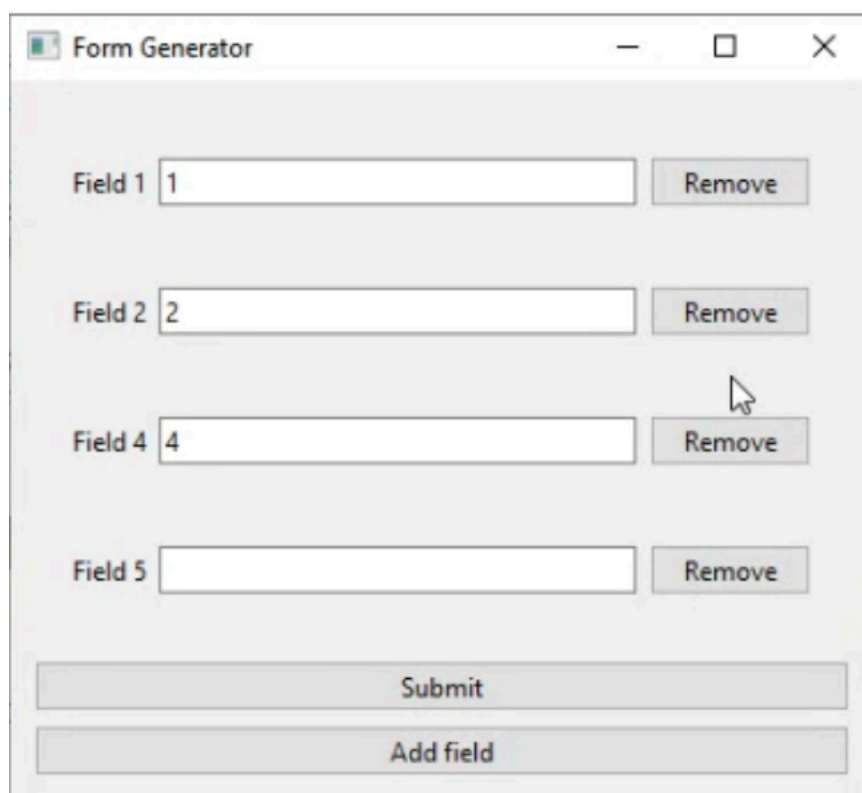
The screenshot shows a window titled "Form Generator". Inside, there are four input fields labeled "Field 1", "Field 2", "Field 4", and "Field 5". Each field contains a number (1, 2, 4, and an empty field respectively). To the right of each field is a "Remove" button. A mouse cursor is pointing at the "Remove" button for "Field 4". At the bottom of the window, there are two large buttons: "Submit" and "Add field".

Рисунок 5 – Проверка удаления полей, удалили Field 3

Нажимаем submit, виджет на данном этапе завершает работу. Результат представлен на рисунке 6.

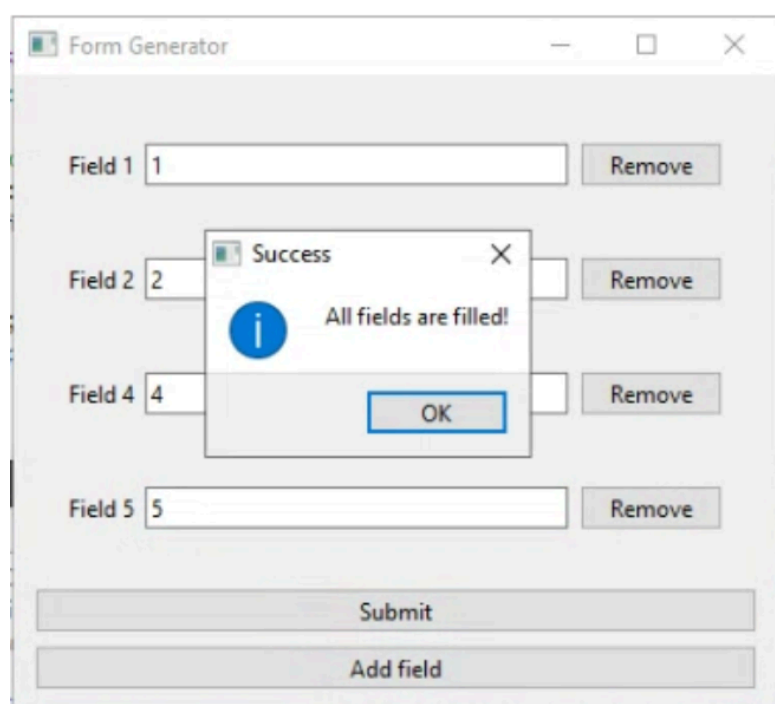
The screenshot shows the same "Form Generator" window, but now there are five fields labeled "Field 1", "Field 2", "Field 4", and "Field 5". Each field contains a number (1, 2, 4, and 5 respectively). A "Success" dialog box is overlaid on the window, displaying an information icon and the text "All fields are filled!". The dialog box has an "OK" button. The "Remove" buttons are still present for each field. The "Submit" and "Add field" buttons are at the bottom.

Рисунок 6 – Виджет завершил работу

Для создания и подключения библиотеки-виджета сначала нужно запустить Qt и создать новый проект с типом "библиотека C++". В настройках проекта выбирается qmake, а модуль core заменяется на gui. Затем внутри .pro-файла для виджета изменяют gui на gui widgets. Можно добавить файл интерфейса .ui, чтобы облегчить работу с графическими элементами. Для этого щелкают правой кнопкой мыши по папке проекта и выбирают добавление нового класса или header-файла. Важно дать правильное имя классу, так как по умолчанию будет использовано Form, что может создать трудности в будущем. При создании лучше оставить все предложенные опции, чтобы файл автоматически добавился в проект через qmake.

После этого нужно реализовать виджет, выполнить сборку или запустить проект. Если в проекте нет файла main, запуск невозможен. Для тестирования виджета внутри текущего проекта создается новый C++ файл main.cpp. Как правило, он автоматически добавляется в .pro-файл, но это стоит проверить. Если файл не добавлен, нужно вручную прописать его в .pro-файле, указав после mainwindow.cpp и добавив обратный слэш (\). Внутри main.cpp указываются заголовочные файлы виджета, создается тестовый объект виджета, и можно попробовать запустить приложение. Если появится ошибка, нужно проверить .pro-файл и убедиться, что там стоит "app" вместо "lib" для тестирования.

Чтобы подключить виджет к другому проекту, после сборки проекта (с установленным модификатором lib в .pro) нужно перейти в директорию сборки, в папке build -> debug скопировать два файла: .a и .dll. Затем создается обычный проект для приложения (qtapplication qmake), и на уровне с ним в корневой директории размещается папка lib, в которой создаются три папки: release, debug и include. В первые две копируются библиотеки, а в include — заголовочные файлы виджета.

После этого в проекте правой кнопкой мыши кликают по папке проекта и добавляют внешнюю библиотеку. Сначала указывают путь к .a файлу (лучше из debug, но можно и из release), затем путь к папке include, после чего лучше отключить совместимость с системами. Далее выполняется компоновка, создается новый виджет, который наследует функционал от вашего виджета. Затем нужно преобразовать его в собственный класс виджета, указав правильное имя класса. В main создается объект вашего виджета, и после этого можно запустить проект — виджет будет готов к использованию. Результат представлен на рисунке 7.

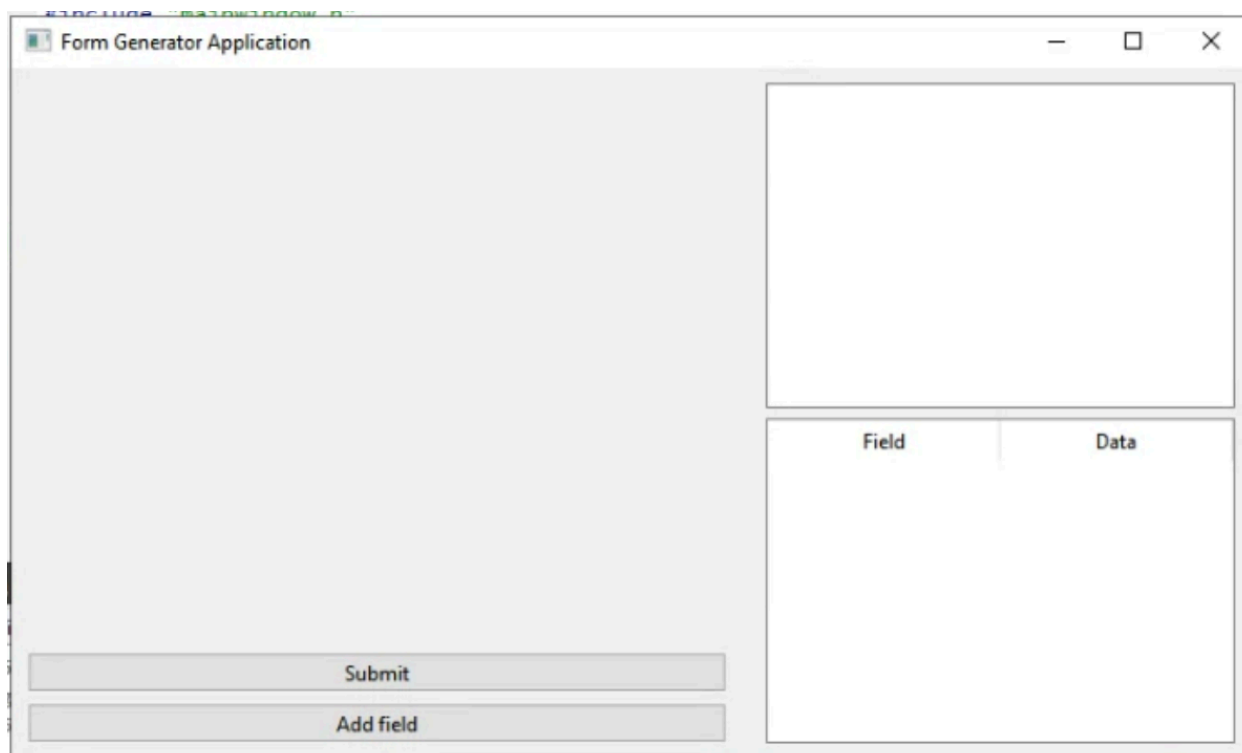


Рисунок 7 – Запуск приложения с созданным виджетом
Заполнение полей анкеты. Результат представлен на рисунке 8.

Field 1

Field 2

Field 3

Field 4

Field	Data

Рисунок 8 – Заполнение полей анкеты

Вывод данных виджета. Результат представлен на рисунке 9.

Field 1

Field 2

Field 3

Field 4

Field 1: Sina
Field 2: Rosa
Field 3: Maria
Field 4: Sea

	Field	Data
1	Field 1	Sina
2	Field 2	Rosa
3	Field 3	Maria
4	Field 4	Sea

Рисунок 9 – Вывод данных виджета в других программных компонентах