ЛАБОРАТОРНАЯ РАБОТА №4

«ИССЛЕДОВАНИЕ СПОСОБОВ ПОСТРОЕНИЯ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ С ПОМОЩЬЮ ЯЗЫКА РАЗМЕТКИ QML»

1. Цель работы

Изучить основы языка разметки QML. Приобрести практические навыки создания графических интерфейсов Qt-приложений на основе разметки.

2. Постановка задачи

2.1. Изучить принципы построения приложений на основе шаблона MVC (выполняется в ходе самостоятельной подготовки к лабораторной работе).

2.2. Выполнить проектирование графического интерфейса приложения, согласно варианту задания (выполняется в ходе самостоятельной подготовки к лабораторной работе).

2.3. Создать проект Qt Quick 2 приложение.

2.4. Добавить собственный класс Button в проект (см. пример в разделе 2.1). Изменить его поведение под свой вариант, если необходимо.

2.5. Создать в дизайнере интерфейс, согласно варианту задания.

2.6. Реализовать логику приложения средствами QML по варианту.

2.7. Выполнить сравнительное исследование методов построения интерфейса пользователя, рассмотренных в лабораторной работе №3, и на основе QML. Сравнение провести по критериям: 1) трудоемкости реализации; 2) гибкости получаемого программного решения.

3. Вариант задания

Вариант задания 2. Внешний вид приложения показан на рисунке 1. Изначально все кнопки, кроме первой, должны быть синего цвета. Первая – красного. При нажатии на кнопку красного цвета, она должна изменить свой цвет на синий. При этом любая другая случайно выбранная кнопка должна изменить свой цвет на красный.

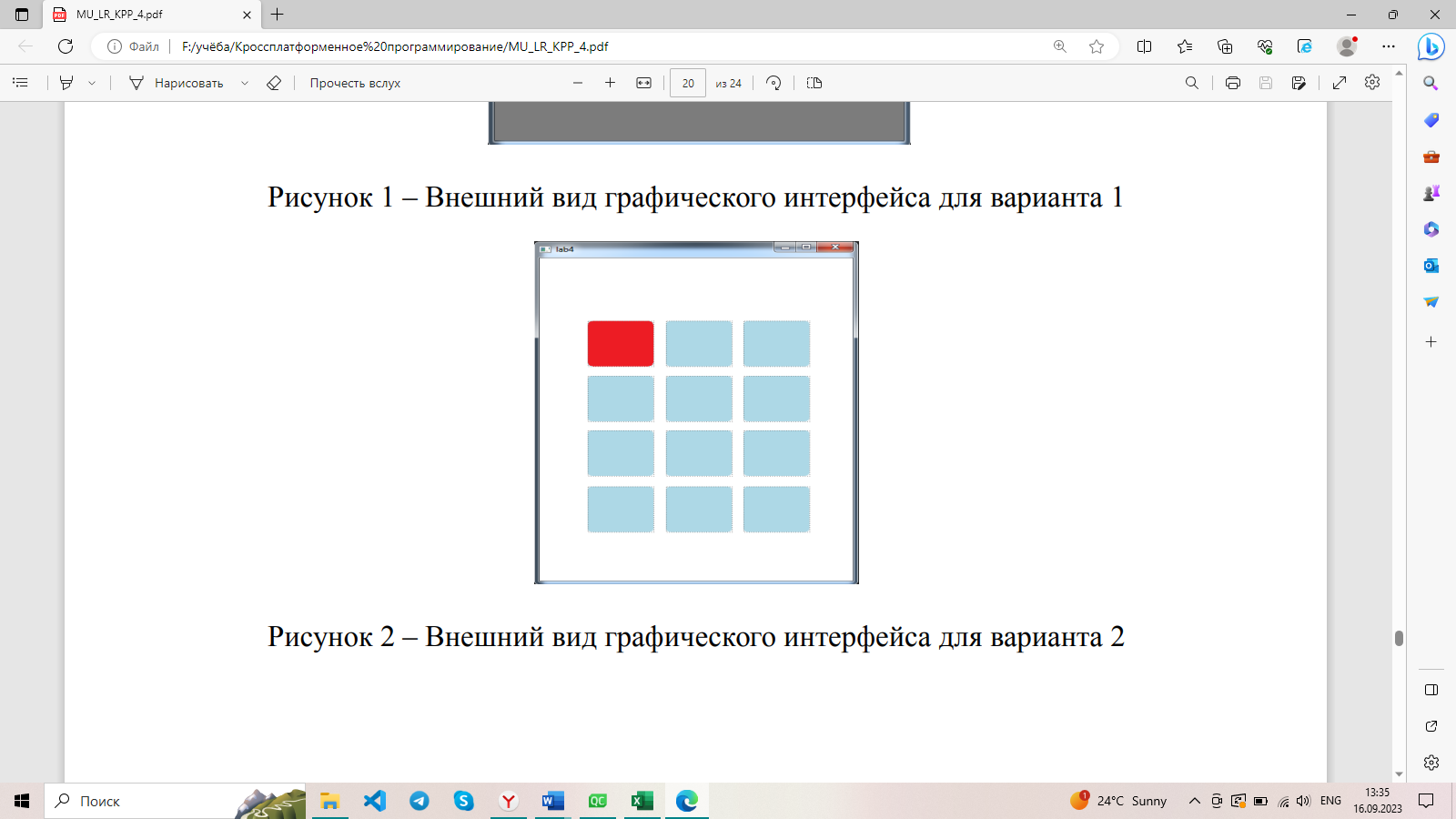


Рисунок 1 - Внешний вид графического интерфейса для варианта 2

4. Ход работы

Был создан новый проект, в который был добавлен файл QML (Qt Quick 2). Процесс добавления файла представлен на рисунке 2.

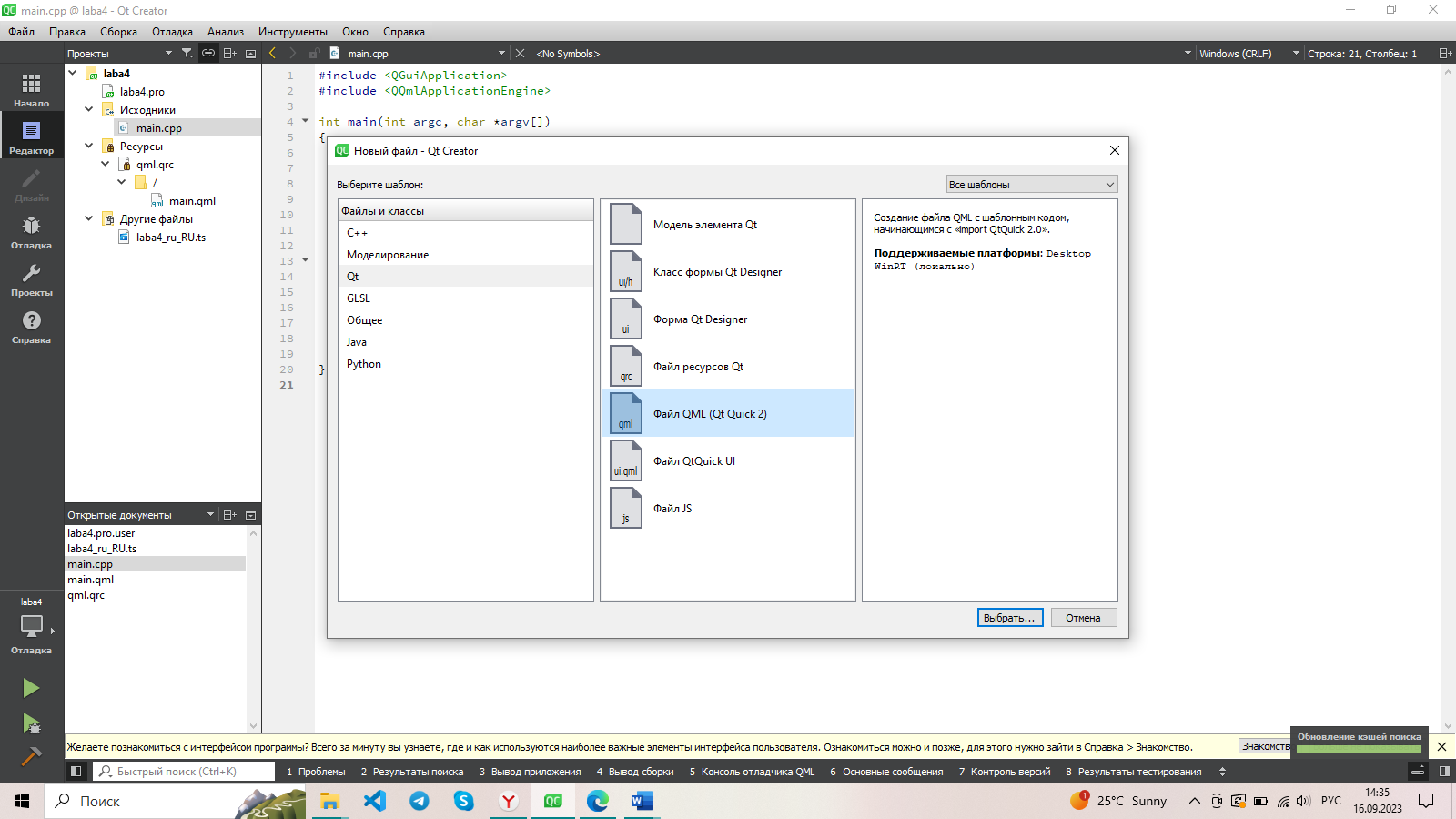


Рисунок 2 – Добавление файла Qt Quick 2

В QML-файле были добавлены кнопки, которые были описаны в файле Button.qml. Интерфейс приложения в окне дизайна представлен на рисунке 3.

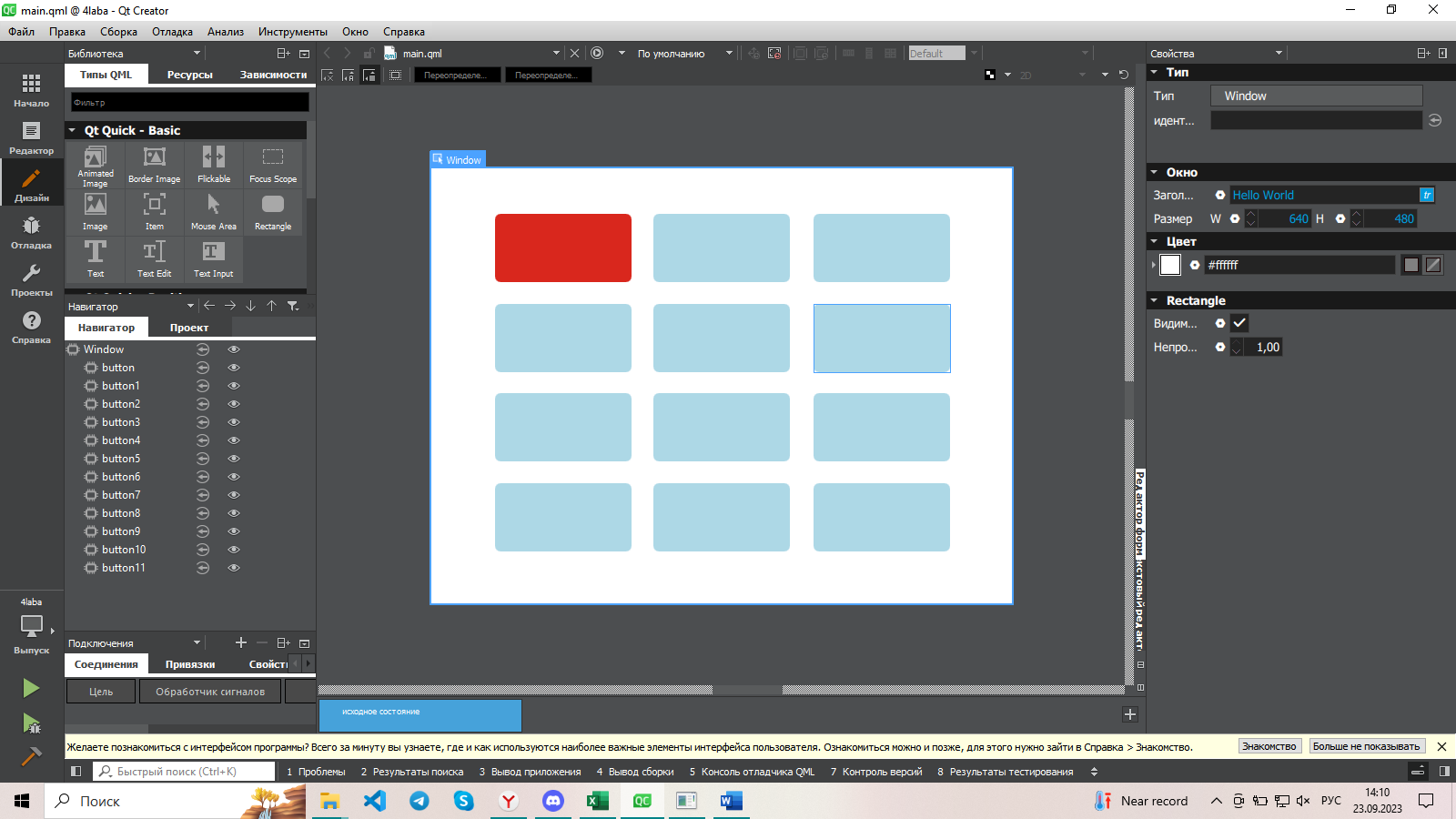


Рисунок 3 – Расположение кнопок и их начальное состояние

Приложение было собрано и запущено. На рисунках 4-5 представлена работа приложения.

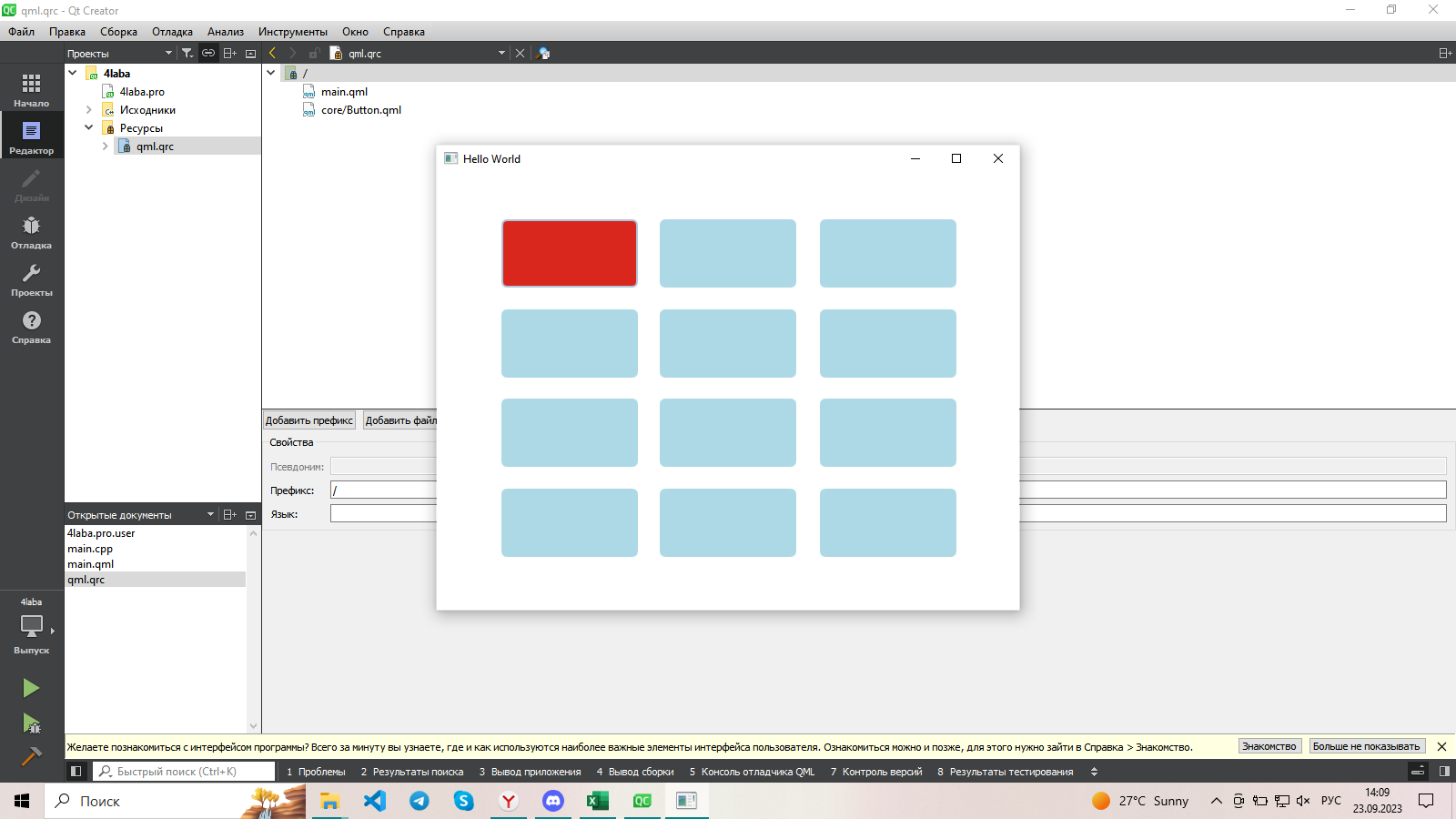


Рисунок 4 – Начальное состояние запущенного приложения

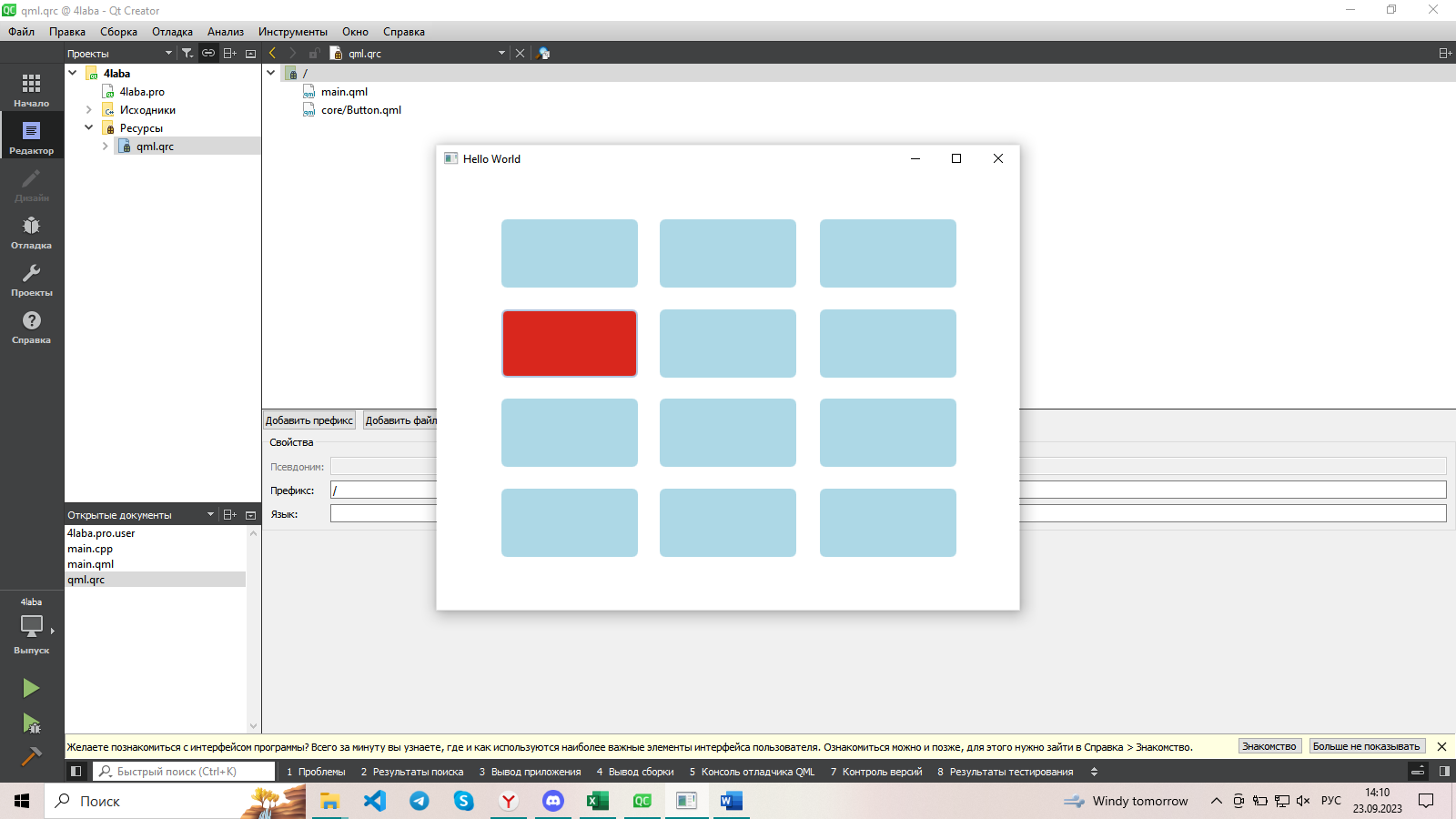


Рисунок 5 – Измененное состояние приложения

В листингах 1-2 представлен код, необходимый для реализации приведенного выше функционала.

Листинг 1 – Код Button.qml

import QtQuick 2.0

Rectangle {

//идентификатор элемента

id: *button*

//эти свойства используются как константы, доступны также

//извне

property int buttonHeight: 75

property int buttonWidth: 150

//свойство для хранения текста на кнопке

property string label

property color textColor: *buttonLabel*.color

//цвет при наведении курсора

property color onHoverColor: "lightsteelblue"

property color borderColor: "transparent"

//цвет кнопки

property color buttonColor: "#add8e6"

property real labelSize: 14

//свойства отображения

radius: 6

property alias buttonLabel: *buttonLabel*

antialiasing: true

border { width: 2; color: *borderColor* }

width: *buttonWidth*; height: *buttonHeight*

Text {

id: *buttonLabel*

anchors.centerIn: *parent*

text: *label* //привязываем текст к тексту

//"родителя"

color: "#000000"

font.pointSize: *labelSize*

}

//сигнал, который будет вызываться при нажатии

signal buttonClick()

//определяем "кликабельную" зону равную поверхности всей

//кнопки

MouseArea {

id: *buttonMouseArea*

anchors.fill: *parent* //размер равен размеру

//"родителя"

onClicked: *buttonClick*()

//если равно true, то будет вызываться onEntered и

//onExited при наведении/удалении курсора

//false - необходимо кликнуть чтобы отработал mouse

//hover

hoverEnabled: true

//отобразить рамку, если навели курсор

onEntered: *parent*.border.color = *onHoverColor*

//удалить рамку при удалении курсора с кнопки

onExited: *parent*.border.color = *borderColor*

}

//изменить цвет кнопки при нажатии

color: *buttonMouseArea*.pressed ? *Qt*.darker(*buttonColor*,

1.5) : *buttonColor*

//анимация смены цвета

Behavior on color { ColorAnimation{ duration: 55 } }

//увеличить кнопку при нажатии

scale: *buttonMouseArea*.pressed ? 1.1 : 1.00

//анимация при увеличении кнопки

Behavior on scale { NumberAnimation{ duration: 55 }}

}

Листинг 2 – Код main.qml

import QtQuick 2.12

import QtQuick.Window 2.12

import "core"

Window {

visible: true

width: 640

height: 480

title: *qsTr*("Hello World")

Button {

id: *button*

x: 71

y: 51

color: "#d9271d"

buttonColor: "#d9271d"

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button*.color == "#d9271d"){

*button*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button*.color = "#d9271d";

}

}

}

}

Button {

id: *button1*

x: 245

y: 51

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button1*.color == "#d9271d"){

*button1*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button1*.color = "#d9271d";

}

}

}

}

Button {

id: *button2*

x: 421

y: 51

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button2*.color == "#d9271d"){

*button2*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button2*.color = "#d9271d";

}

}

}

}

Button {

id: *button3*

x: 71

y: 150

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button3*.color == "#d9271d"){

*button3*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button3*.color = "#d9271d";

}

}

}

}

Button {

id: *button4*

x: 245

y: 150

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button4*.color == "#d9271d"){

*button4*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button4*.color = "#d9271d";

}

}

}

}

Button {

id: *button5*

x: 421

y: 150

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button5*.color == "#d9271d"){

*button5*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button5*.color = "#d9271d";

}

}

}

}

Button {

id: *button6*

x: 71

y: 248

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button6*.color == "#d9271d"){

*button6*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button6*.color = "#d9271d";

}

}

}

}

Button {

id: *button7*

x: 245

y: 248

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button7*.color == "#d9271d"){

*button7*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button7*.color = "#d9271d";

}

}

}

}

Button {

id: *button8*

x: 421

y: 248

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button8*.color == "#d9271d"){

*button8*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button8*.color = "#d9271d";

}

}

}

}

Button {

id: *button9*

x: 71

y: 347

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button9*.color == "#d9271d"){

*button9*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button9*.color = "#d9271d";

}

}

}

}

Button {

id: *button10*

x: 245

y: 347

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button10*.color == "#d9271d"){

*button10*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button10*.color = "#d9271d";

}

}

}

}

Button {

id: *button11*

x: 421

y: 347

onButtonClick: {

// Изменение цвета красной кнопки на синий

if(*button11*.color == "#d9271d"){

*button11*.color = "#add8e6";

// Случайный выбор другой кнопки

var *randomIndex* = *Math*.floor(*Math*.random() \* *parent*.children.length);

var *randomButton* = *parent*.children[*randomIndex*];

if (*randomButton* !== this) {

*randomButton*.color = "#d9271d";

}

else {

*button11*.color = "#d9271d";

}

}

}

}

}

ВЫВОДЫ

В ходе выполнения лабораторной работы были изучены основные методики создания графического пользовательского интерфейса с использованием виджетов QtWidgets. Было создано приложение с помощью Qt Designer в соответствии с заданием по варианту, а также разработан функционал, а именно валидация форм. В ходе разработки программного кода проводилось тестирование функции валидации, а по результатам тестов установлено, что функция работает корректно.

Контрольные вопросы:

5.1. Что такое QML?

QML (Qt Meta-Object Language) - это декларативный язык разметки, используемый в Qt для описания пользовательского интерфейса и его взаимодействия с логикой приложения. QML позволяет создавать гибкие и интерактивные пользовательские интерфейсы, упрощая проектирование и разработку.

5.2. В чем состоит различие декларативного и императивного подхода?

Декларативный подход (как в QML) фокусируется на описании "что" должно происходить, без явного указания "как" это должно быть достигнуто. Он использует декларативные языки и конструкции для описания свойств, структуры и взаимодействия элементов системы. Программист описывает желаемое состояние системы, а исполнение логики происходит автоматически.

Императивный подход фокусируется на описании последовательности команд и действий, которые должны быть выполнены для достижения определенного результата. Программист явно указывает порядок выполнения операций и контролирует каждый шаг.

5.3. Какие плюсы в разделении частей логики и представления в приложении?

Улучшенная читаемость и поддерживаемость кода: Разделение логики и представления позволяет легче понять и изменять каждую часть независимо от другой. Это способствует улучшению поддержки кода и упрощению сопровождения приложения.

Повторное использование: Отделение логики от представления делает возможным повторное использование одной и той же логики с различными представлениями. Это позволяет создавать гибкие и масштабируемые приложения.

Улучшенная параллельность работы: Разделение логики и представления позволяет разработчикам лучше распределить работу между различными командами и параллельно работать над разными аспектами приложения.

5.4. Какие минусы в разделении частей логики и представления в приложении?

Усложнение коммуникации: Разделение может привести к необходимости установления механизмов коммуникации между логикой и представлением, что может добавить сложности в разработку и поддержку.

Дополнительный объем кода: Разделение логики и представления может привести к необходимости написания дополнительного кода для связи между ними, что может увеличить объем кода приложения.

Усложнение отладки: Разделение может затруднить отладку приложения, поскольку ошибки могут быть связаны с несоответствием между логикой и представлением.

5.5. На каком языке пишется интерактивная часть приложений QML?

Интерактивная часть приложений QML пишется на языке JavaScript. JavaScript используется для определения поведения элементов интерфейса, обработки событий и взаимодействия с логикой приложения.

5.6. Какой паттерн проектирования рекомендуется к использованию с QML?

С QML рекомендуется использовать паттерн проектирования Model-View-Controller (MVC). MVC разделяет приложение на три компонента: модель (Model), представление (View) и контроллер (Controller). Модель отвечает за предоставление данных, представление отображает данные пользователю, а контроллер обрабатывает пользовательский ввод и управляет взаимодействием между моделью и представлением.

5.7. На основе какого формата построен язык разметки QML?

Язык разметки QML построен на основе формата JSON (JavaScript Object Notation). Он использует синтаксис, основанный на JavaScript, для определения структуры элементов интерфейса, иерархии и свойств.

5.8. Каково назначение контроллеров в шаблоне MVC?

В шаблоне MVC контроллеры отвечают за обработку пользовательского ввода, управление моделью и обновление представления. Они служат посредниками между моделью и представлением, обеспечивая взаимодействие и согласованность данных между ними.

5.9. Каково назначение моделей в шаблоне MVC?

Модели в шаблоне MVC представляют данные, с которыми работает приложение. Они отвечают за доступ к данным, их обновление и предоставление информации представлениям и контроллерам. Модели могут содержать бизнес-логику, а также методы для получения и обновления данных.

5.10. Каково назначение представлений в шаблоне MVC?

Представления в шаблоне MVC отображают данные модели и обрабатывают пользовательский ввод. Они отвечают за отображение информации пользователю и обновление интерфейса в соответствии с изменениями в модели. Представления могут быть графическими элементами, такими как окна, кнопки, таблицы и т.д.

5.11. Для чего применяются делегаты в MVC-приложениях?

В MVC-приложениях делегаты используются для управления отображением и редактированием отдельных элементов данных в представлении. Делегаты позволяют настраивать способы отображения и редактирования элементов, а также обрабатывать события, связанные с элементами данных. Они обеспечивают гибкость в настройке внешнего вида и поведения элементов представления в зависимости от данных и контекста приложения.