//main.cpp

#include "menuwindow.h"

#include <QApplication>

#include <QFile>

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

{

QApplication a(argc, argv);

//Set the app style sheet

QFile styleSheetFile("../GUITheme/SyNet.qss");

styleSheetFile.open(QFile::ReadOnly);

QString styleSheet = QLatin1String(styleSheetFile.readAll());

a.setStyleSheet(styleSheet);

MenuWindow w;

w.show();

return a.exec();

}

//menuwindow.cpp

#include "menuwindow.h"

#include "ui\_menuwindow.h"

#include "dialogforcreatingform.h"

//This code initializes the MenuWindow constructor and establishes

//connections between different signals and slots to handle events in the user interface.

MenuWindow::MenuWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MenuWindow)

{

ui->setupUi(this); // Setting up the user interface

// Creating objects

testWidget = new TestWidget;

testForm = new CreatingTestForm;

scroll = new HorizontalScrollArea(3, 3);

// Adding the scroll to the tests\_page in the user interface layout

ui->tests\_page->layout()->addWidget(scroll);

// Connecting signals to slots

connect(ui->testsPushButton, &QPushButton::clicked, this, &MenuWindow::creating\_Tests\_page);

// Connecting signals from objects to slots in this MenuWindow

connect(testWidget, &TestWidget::open\_menuwindow, this, &MenuWindow::show);

connect(testForm, &CreatingTestForm::open\_menuwindow, this, &MenuWindow::show);

connect(testForm, &CreatingTestForm::open\_menuwindow, this, &MenuWindow::creating\_Tests\_page);

connect(this, &MenuWindow::open\_menuwindow, this, &MenuWindow::creating\_Tests\_page);

}

MenuWindow::~MenuWindow()

{

delete ui;

}

//The slot generated by the Qt environment for the "Tests" button sets the next widget in the

//StackedWidgets and calls the function that generates the current test directory page.

void MenuWindow::on\_testsPushButton\_clicked()

{

ui->stackedWidget->setCurrentWidget(ui->tests\_page);

creating\_Tests\_page();

}

//When a specific test is clicked, the current window closes and the selected test opens

void MenuWindow::dynamicPushButton\_clicked()

{

this->close();

testWidget->show();

}

//This function generates the current test directory,dynamically creating buttons

//by taking names from the test directory. For these buttons, a class QRightClickButton

//was created, which contains the following signals: shift + right click; doubleRightClicked, leftClicked.

void MenuWindow::creating\_Tests\_page()

{

for(QRightClickButton\* i : dynamic\_buttons)

{

delete i;

}

dynamic\_buttons.clear();

const QSize btnSize = QSize(400, 100);

QStringList list\_dir = count\_files\_in\_dir("..\\TestControl\\alltests");

int counter = 0;

bool flag = true; //It is used to determine in which row/column to add to our scroll (from the HorizontalScrollArea class)

for(QString i : list\_dir)

{

QRightClickButton\* newButton = new QRightClickButton(ui->tests\_page);

newButton->setText(i);

newButton->setSizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);

dynamic\_buttons.append(newButton);

// Establishes connections for different button click events

// Left-click on newButton connected to 'setPath' method in 'testWidget'

// Left-click on newButton connected to 'dynamicPushButton\_clicked' method in 'MenuWindow'

// Shift + Right-click on newButton connected to 'shift\_right\_dynamicPushButton\_clicked' method in 'MenuWindow'

// Double right-click on newButton connected to 'double\_right\_dynamicPushButton\_clicked' method in 'MenuWindow'

connect(newButton, &QRightClickButton::leftClicked,this->testWidget, &TestWidget::setPath);

connect(newButton, &QRightClickButton::leftClicked, this, &MenuWindow::dynamicPushButton\_clicked);

connect(newButton, &QRightClickButton::shiftRightClicked, this, &MenuWindow::shift\_right\_dynamicPushButton\_clicked);

connect(newButton, &QRightClickButton::doubleRightClicked, this, &MenuWindow::double\_right\_dynamicPushButton\_clicked);

if(flag)

{

scroll->addWidget(newButton,0, counter);

flag = false;

continue;

}

if(!flag)

{

scroll->addWidget(newButton,1, counter);

flag = true;

}

counter++;

}

}

//This code handles the event when the "Create" button is clicked.

//It opens a dialog box for user input. If the user clicks "Accept" in the dialog,

//the program creates a new directory for a test using the entered name. If a directory with the same name

//already exists, it displays a warning.

//In either case, it either creates a new test or rejects the user input.

void MenuWindow::on\_createPushButton\_clicked()

{

DialogForCreatingForm dlg( this );

switch( dlg.exec() ) {

case QDialog::Accepted:{

connect(this, &MenuWindow::open\_creating\_tests\_form, this->testForm, &CreatingTestForm::generate\_creating\_form);

QDir dir(QString("..\\TestControl\\alltests\\%1\\").arg(dlg.getInput()));

QFile file(QString("..\\TestControl\\alltests\\%1\\%1.txt").arg(dlg.getInput().arg(dlg.getInput())));

if(!dir.exists())

{

qDebug() << dlg.getInput();

qDebug() << "Accepted";

dir.setPath("..\\TestControl\\alltests\\");

testForm->set\_path\_to\_test(QString("..\\TestControl\\alltests\\%1\\").arg(dlg.getInput()));

testForm->set\_name\_of\_test(QString("%1.txt").arg(dlg.getInput()));

dir.mkdir(dlg.getInput());

emit open\_creating\_tests\_form();

file.open(QIODevice::ReadWrite);

file.close();

this->close();

testForm->show();

}

else

{

QMessageBox::warning(this, "TestControl","The same name of the test already exist!");

}

break;

}

case QDialog::Rejected:{

qDebug() << "Rejected";

break;

}

default:{

qDebug() << "Unexpected";

}

}

}

//Deletes the current test by removing it from the directory and

//emits the signal creating\_Tests\_page, which triggers the reloading of the tests directory.

void MenuWindow::shift\_right\_dynamicPushButton\_clicked()

{

QMessageBox::StandardButton reply = QMessageBox::question(this, "TestControl", "Are you sure?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::Yes)

{

QRightClickButton\* sender\_button = qobject\_cast<QRightClickButton\*>(sender());

QString path = QString("..\\TestControl\\alltests\\%1\\").arg(sender\_button->text());

QDir dir(path);

dir.removeRecursively();

emit creating\_Tests\_page();

}

}

// Responds to a double right-click event on a QRightClickButton

// Retrieves the sender button and constructs the path and name for the selected test

// Sets the path and name of the test in the 'testForm' object

// Switches 'testForm' to the creating form for editing a slide

// Closes the current window and shows the 'testForm'

void MenuWindow::double\_right\_dynamicPushButton\_clicked()

{

QRightClickButton\* sender\_button = qobject\_cast<QRightClickButton\*>(sender());

QString path\_to\_test = QString("..\\TestControl\\alltests\\%1\\").arg(sender\_button->text());

QString name\_of\_test = QString("%1.txt").arg(sender\_button->text());

testForm->set\_path\_to\_test(path\_to\_test);

testForm->set\_name\_of\_test(name\_of\_test);

testForm->setCreatingFormForEditSlide();

this->close();

testForm->show();

}

void MenuWindow::on\_exitPushButton\_clicked()

{

QApplication::exit();

}

//creatingtestform.cpp

#include "creatingtestform.h"

#include "ui\_creatingtestform.h"

CreatingTestForm::CreatingTestForm(QWidget \*parent) :

QWidget(parent),

ui(new Ui::CreatingTestForm)

{

ui->setupUi(this);

generate\_image();

connect(ui->cancelPushButton, SIGNAL(clicked(bool)), this, SLOT(cancelPushButton\_clicked()));

}

void CreatingTestForm::set\_path\_to\_test(const QString& path)

{

path\_to\_test = path;

}

void CreatingTestForm::set\_name\_of\_test(const QString &name)

{

name\_of\_test = name;

}

CreatingTestForm::~CreatingTestForm()

{

delete ui;

}

void CreatingTestForm::generate\_creating\_form()

{

ui->numOfSlideLabel->setText(QString("Slide №") + QString("%1").arg(num\_of\_slide + 1));

create\_current\_info\_map(myTest, num\_of\_slide);

ui->justQuestionLineEdit->setText(myTest[num\_of\_slide]["question"]);

imgLabel->setCurrentIMG(path\_to\_IMG);

}

void CreatingTestForm::setCreatingFormForEditSlide()

{

myTest = parse\_txt\_to\_test(path\_to\_test + name\_of\_test);

is\_chosen = true;

is\_next = true;

num\_of\_slide = 0;

ui->numOfSlideLabel->setText(QString("Slide №") + QString("%1").arg(num\_of\_slide + 1));

path\_to\_IMG = myTest[num\_of\_slide]["img"];

imgLabel->setCurrentIMG(path\_to\_IMG);

ui->justQuestionLineEdit->setText(myTest[num\_of\_slide]["question"]);

is\_editor = true;

}

void CreatingTestForm::cancelPushButton\_clicked()

{

QMessageBox::StandardButton reply = QMessageBox::question(this, "TestControl", "Are you sure?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::Yes)

{

if(!is\_editor)

{

QDir dir(path\_to\_test);

dir.removeRecursively();

}

num\_of\_slide = 0;

num\_of\_right\_answer = "NONE";

is\_chosen = false;

is\_next = false;

is\_editor = false;

path\_to\_test = "";

name\_of\_test = "";

// "IMG\_URL|Question|num\_of\_questions|number\_of\_right\_question|question1/question2/../questionN|"

info\_in\_text\_file = "%1|%2|%3|%4|%5";

myTest.clear();

buttons\_list.clear();

this->close();

emit open\_menuwindow();

}

}

void CreatingTestForm::on\_savePushButton\_clicked()

{

if(is\_avaiable\_to\_write\_in\_txt\_file(myTest))

{

put\_map\_into\_file(myTest, path\_to\_test + name\_of\_test);

num\_of\_slide = 0;

num\_of\_right\_answer = "NONE";

is\_chosen = false;

is\_next = false;

is\_editor = false;

path\_to\_test = "";

name\_of\_test = "";

// "IMG\_URL|Question|num\_of\_questions|number\_of\_right\_question|question1/question2/../questionN|"

info\_in\_text\_file = "%1|%2|%3|%4|%5";

myTest.clear();

buttons\_list.clear();

this->close();

emit open\_menuwindow();

}

else

{

QMessageBox::warning(this, "TestControl","Add questions everywhere!");

}

}

void CreatingTestForm::on\_editQuestionsPushButton\_clicked()

{

num\_of\_right\_answer = myTest[num\_of\_slide]["number\_of\_right\_answer"];

ui->stackedWidget->setCurrentWidget(ui->addAnswersWidget);

QStringList list = parse\_line\_into\_questions(myTest[num\_of\_slide]["variants\_of\_answers"]);

for(QString button\_text : list)

{

if(button\_text != "" && button\_text != "NONE"){

QRightClickButton\* new\_button = new QRightClickButton(ui->addAnswersWidget);

new\_button->setText(button\_text);

new\_button->setSizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);

connect(new\_button, &QRightClickButton::shiftRightClicked, this, &CreatingTestForm::onRemovedButton);

connect(new\_button, &QRightClickButton::leftClicked, this, &CreatingTestForm::chooseRightAnswer);

buttons\_list.append(new\_button);

ui->verticalLayout\_forQuestions->insertWidget(ui->verticalLayout\_forQuestions->count(),new\_button);

}

}

if(!buttons\_list.isEmpty() && num\_of\_right\_answer != "NONE")

{

QRightClickButton\* button = buttons\_list[num\_of\_right\_answer.toInt() - 1];

button->setStyleSheet("background-color: green");

}

}

void CreatingTestForm::on\_cancelPushButton\_2\_clicked()

{

ui->stackedWidget->setCurrentWidget(ui->creatingMenuWidget);

for(QRightClickButton\* i : buttons\_list)

{

delete i;

}

buttons\_list.clear();

is\_chosen = false;

}

void CreatingTestForm::create\_current\_info\_map(QMap<int, QMap<QString, QString>>& tmp\_map,int num\_of\_slide)

{

if(tmp\_map[num\_of\_slide].isEmpty())

{

QMap<QString, QString> tmp\_slide\_map;

tmp\_slide\_map["img"] = "NONE";

tmp\_slide\_map["question"] = "NONE";

tmp\_slide\_map["amount\_of\_questions"] = "NONE";

tmp\_slide\_map["number\_of\_right\_answer"] = "NONE";

tmp\_slide\_map["variants\_of\_answers"] = "NONE";

tmp\_map[num\_of\_slide] = tmp\_slide\_map;

is\_next = false;

}

num\_of\_right\_answer = myTest[num\_of\_slide]["number\_of\_right\_answer"];

is\_next = is\_avaiable\_nextSlidePushButton();

is\_chosen = is\_next;

path\_to\_IMG = myTest[num\_of\_slide]["img"];

}

bool CreatingTestForm::is\_avaiable\_nextSlidePushButton()

{

QString question, amount\_of\_questions, number\_of\_right\_answer, variants\_of\_answer;

question = myTest[num\_of\_slide]["question"];

amount\_of\_questions = myTest[num\_of\_slide]["amount\_of\_questions"];

number\_of\_right\_answer = myTest[num\_of\_slide]["number\_of\_right\_answer"];

variants\_of\_answer = myTest[num\_of\_slide]["variants\_of\_answer"];

if(question != "NONE" && amount\_of\_questions != "NONE" && number\_of\_right\_answer != "NONE" && variants\_of\_answer != "NONE")

{

return true;

}

else

{

return false;

}

}

void CreatingTestForm::on\_addQuestionPushButton\_clicked()

{

QString button\_text = ui->questionLineEdit->text();

QRightClickButton\* new\_button = new QRightClickButton(ui->addAnswersWidget);

new\_button->setText(button\_text);

new\_button->setSizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);

connect(new\_button, &QRightClickButton::shiftRightClicked, this, &CreatingTestForm::onRemovedButton);

connect(new\_button, &QRightClickButton::leftClicked, this, &CreatingTestForm::chooseRightAnswer);

buttons\_list.append(new\_button);

ui->verticalLayout\_forQuestions->insertWidget(ui->verticalLayout\_forQuestions->count(),new\_button);

ui->questionLineEdit->setText("");

}

void CreatingTestForm::onRemovedButton()

{

QRightClickButton\* button = qobject\_cast<QRightClickButton\*>(sender());

buttons\_list.removeOne(button);

delete button;

}

QString CreatingTestForm::get\_number\_of\_right\_answer(const QList<QRightClickButton\*>& list)

{

int counter = 1;

for(QRightClickButton\* btn : list)

{

if(btn->styleSheet() == "background-color: green")

{

return QString("%1").arg(counter);

}

counter++;

}

if(counter >= list.size())

return QString("NONE");

}

void CreatingTestForm::on\_savePushButton\_2\_clicked()

{

if(ui->justQuestionLineEdit->text() == "")

{

myTest[num\_of\_slide]["question"] = "NONE";

}

else

{

myTest[num\_of\_slide]["question"] = ui->justQuestionLineEdit->text();

}

myTest[num\_of\_slide]["amount\_of\_questions"] = QString("%1").arg(buttons\_list.size());

myTest[num\_of\_slide]["number\_of\_right\_answer"] = get\_number\_of\_right\_answer(buttons\_list);

QString variants\_of\_answers;

for(QRightClickButton\* i : buttons\_list)

{

variants\_of\_answers += i->text() + '/';

delete i;

}

variants\_of\_answers.removeAt(variants\_of\_answers.size()-1);

buttons\_list.clear();

myTest[num\_of\_slide]["variants\_of\_answers"] = variants\_of\_answers;

ui->stackedWidget->setCurrentWidget(ui->creatingMenuWidget);

is\_next = is\_avaiable\_nextSlidePushButton();

}

void CreatingTestForm::chooseRightAnswer()

{

QRightClickButton\* button = qobject\_cast<QRightClickButton\*>(sender());

int counter = 1;

if(!is\_chosen)

{

for(QRightClickButton\* b : buttons\_list)

{

if(b == button){

num\_of\_right\_answer = QString("%1").arg(counter);

break;

}

counter++;

}

button->setStyleSheet("background-color: green");

is\_chosen = true;

return;

}

else if(num\_of\_right\_answer != "NONE")

{

if(buttons\_list[num\_of\_right\_answer.toInt() - 1] == button && is\_chosen)

{

button->setStyleSheet("");

is\_chosen = false;

num\_of\_right\_answer = "NONE";

}

return;

}

}

void CreatingTestForm::on\_nextSlideButton\_clicked()

{

if(is\_next)

{

num\_of\_slide++;

generate\_creating\_form();

}

else

{

QMessageBox::warning(this, "TestControl","This slide isn't created yet");

}

}

void CreatingTestForm::on\_prevPushButton\_clicked()

{

if(num\_of\_slide > 0)

{

num\_of\_slide--;

generate\_creating\_form();

}

}

void CreatingTestForm::on\_addImagePushButton\_clicked()

{

imgLabel->is\_changed = false;

ui->stackedWidget->setCurrentWidget(ui->addImageWidget);

}

void CreatingTestForm::generate\_image()

{

QVBoxLayout\* layout = new QVBoxLayout(ui->addImageWidget);

imgLabel = new AspectRatioPixmapLabel;

imgLabel->setText("NONE");

imgLabel->setAlignment(Qt::AlignCenter);

imgLabel->setClicable(true);

QPushButton\* returnPushButton = new QPushButton;

returnPushButton->setText("OK");

connect(returnPushButton, &QPushButton::clicked, this, &CreatingTestForm::returnPushButton\_clicked);

layout->addWidget(imgLabel);

layout->addWidget(returnPushButton);

}

void CreatingTestForm::returnPushButton\_clicked()

{

if(imgLabel->is\_changed){

path\_to\_IMG = imgLabel->getPath\_to\_IMG();

QFileInfo fileInfo(path\_to\_IMG);

QString newFilePath = path\_to\_test + fileInfo.fileName();

if(is\_need\_to\_delete\_IMG(myTest[num\_of\_slide]["img"]))

{

QFile file(myTest[num\_of\_slide]["img"]);

file.remove();

file.close();

}

if (QFile::copy(path\_to\_IMG, newFilePath)) {

qDebug() << "Файл скопирован успешно: " << newFilePath;

} else {

qDebug() << "Не удалось скопировать файл";

}

path\_to\_IMG == "NONE" ? path\_to\_IMG : path\_to\_IMG = newFilePath;

myTest[num\_of\_slide]["img"] = path\_to\_IMG;

imgLabel->is\_changed = false;

}

ui->stackedWidget->setCurrentWidget(ui->creatingMenuWidget);

}

bool CreatingTestForm::is\_need\_to\_delete\_IMG(const QString& path\_to\_img)

{

int counter = 0;

for(QMap<QString, QString> i : myTest)

{

if (i["img"] == path\_to\_img)

counter++;

}

return counter >= 2 ? false : true;

}

void CreatingTestForm::keyPressEvent(QKeyEvent \*e)

{

if(e->key() == Qt::Key\_Delete)

{

QMessageBox::StandardButton reply = QMessageBox::question(this, "TestControl", "Are you sure?", QMessageBox::Yes | QMessageBox::No);

if (reply == QMessageBox::Yes)

{

removeCurrentPage(num\_of\_slide);

generate\_creating\_form();

}

}

}

void CreatingTestForm::removeCurrentPage(int index\_to\_remove)

{

if(index\_to\_remove >= 0 && index\_to\_remove < myTest.size())

{

for(int i = index\_to\_remove; i < myTest.size() - 1; ++i)

{

myTest[i] = myTest[i+1];

}

if(index\_to\_remove == myTest.size() - 1 && index\_to\_remove > 0)

num\_of\_slide--;

myTest.remove(myTest.size() - 1);

}

}

//testwidget.cpp

#include "testwidget.h"

#include "ui\_testwidget.h"

#include <QDebug>

//The code represents the TestWidget widget used to display and interact with tests.

//Here's an explanation of its functionality:

//Constructor and destructor:

//The constructor sets up the widget's interface, while the destructor releases allocated memory.

//Signal and slots: A connection is established between the activate\_generating\_slide()

//signal and the generate\_slide() slot to generate a new slide upon signal invocation.

//Adding answers:

//The add\_answers() function creates buttons with answer options based on the test data.

//Handling the correct answer: The change\_style\_if\_right\_answer() function modifies the button style based on the chosen answer.

//Slide generation:

//The generate\_slide() function refreshes the test slide, including the question, image, and answer options.

//Button control:

//The make\_buttons\_unclickable() and delete\_dynamic\_buttons() functions manage button states during testing.

//Handling keypress events:

//The keyPressEvent() method responds to keypresses, allowing users to use the Enter/Return keys to navigate to the next question and Esc to return to the menu.

TestWidget::TestWidget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::TestWidget)

{

ui->setupUi(this);

connect(this, SIGNAL(activate\_generating\_slide()), this, SLOT(generate\_slide()));

}

TestWidget::~TestWidget()

{

delete ui;

}

void TestWidget::add\_answers()

{

QStringList variants\_of\_answers = parse\_line\_into\_questions(myTest[num\_of\_slide]["variants\_of\_answers"]);

int counter = 0;

bool flag = true;

for(QString i : variants\_of\_answers)

{

QPushButton\* newButton = new QPushButton(this);

newButton->setText(i);

newButton->setSizePolicy(QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding);

button\_vector.append(newButton);

connect(newButton, SIGNAL(clicked(bool)), this, SLOT(change\_style\_if\_right\_answer()));

ui->verticalLayout->addWidget(newButton);

counter++;

}

}

void TestWidget::action\_if\_right\_answer()

{

if(num\_of\_slide == myTest.count()){

QMessageBox::information(this, "TestControl",QString("Result: %1/%2").arg(score).arg(num\_of\_slide));

this->close();

score = 0;

num\_of\_slide = 0;

delete\_dynamic\_buttons();

if(is\_image)

{

ui->splitter->widget(0)->deleteLater();

is\_image = false;

}

emit open\_menuwindow();

}

generate\_slide();

}

void TestWidget::delete\_dynamic\_buttons()

{

for(QPushButton\* i : button\_vector){

ui->verticalLayout->removeWidget(i);

delete(i);

}

button\_vector.clear();

}

void TestWidget::generate\_slide()

{

can\_we\_continue = false;

delete\_dynamic\_buttons();

add\_image\_on\_slide();

add\_answers();

QMap<QString, QString> slide = myTest[num\_of\_slide];

ui->questionLabel->setText(slide["question"]);

}

void TestWidget::add\_image\_on\_slide()

{

imgLabel = new AspectRatioPixmapLabel;

imgLabel->setText("NONE");

QString path\_to\_img = QString(myTest[num\_of\_slide]["img"]);

QPixmap pic(path\_to\_img);

if(is\_image)

{

ui->splitter->widget(0)->deleteLater();

is\_image = false;

}

if(path\_to\_img != "NONE" && !pic.isNull())

{

ui->splitter->insertWidget(0, imgLabel);

imgLabel->setPixmap(pic);

is\_image = true;

}

}

void TestWidget::change\_style\_if\_right\_answer()

{

can\_we\_continue = true;

int number\_of\_right\_answer = (myTest[num\_of\_slide]["number\_of\_right\_answer"]).toInt();

QStringList variants\_of\_answers = parse\_line\_into\_questions(myTest[num\_of\_slide]["variants\_of\_answers"]);

QPushButton\* senderButton = qobject\_cast<QPushButton\*>(sender());

if(variants\_of\_answers[number\_of\_right\_answer-1] == senderButton->text())

{

senderButton->setStyleSheet("background-color: green");

score++;

num\_of\_slide++;

make\_buttons\_unclickable();

}

else

{

QPushButton\* btn = button\_vector[number\_of\_right\_answer-1];

btn->setStyleSheet("background-color: green");

senderButton->setStyleSheet("background-color: red");

num\_of\_slide++;

make\_buttons\_unclickable();

}

}

void TestWidget::make\_buttons\_unclickable()

{

for(QPushButton\* i : button\_vector)

{

i->setEnabled(false);

}

}

void TestWidget::setPath()

{

QPushButton\* senderButton = qobject\_cast<QPushButton\*>(sender());

path\_to\_test = QString("../TestControl/alltests/%1/%1.txt").arg(senderButton->text());

myTest = parse\_txt\_to\_test(path\_to\_test);

emit activate\_generating\_slide();

}

void TestWidget::keyPressEvent(QKeyEvent \*e){

if( (e->key() == Qt::Key\_Enter || e->key() == Qt::Key\_Return ) && can\_we\_continue)

action\_if\_right\_answer();

if(e->key() == Qt::Key\_Escape)

{

this->close();

score = 0;

num\_of\_slide = 0;

delete\_dynamic\_buttons();

if(is\_image)

{

ui->splitter->widget(0)->deleteLater();

is\_image = false;

}

emit open\_menuwindow();

}

}

//aspectratiopixmaplabel.h

#include "aspectratiopixmaplabel.h"

//#include <QDebug>

AspectRatioPixmapLabel::AspectRatioPixmapLabel(QWidget \*parent) :

QLabel(parent)

{

setAcceptDrops(true);

this->setMinimumSize(1,1);

setScaledContents(false);

}

void AspectRatioPixmapLabel::setPixmap ( const QPixmap & p)

{

pix = p;

if(!pix.isNull()){

QLabel::setPixmap(scaledPixmap());

}else{

QLabel::setText("NONE");

}

}

int AspectRatioPixmapLabel::heightForWidth( int width ) const

{

return pix.isNull() ? this->height() : ((qreal)pix.height()\*width)/pix.width();

}

QSize AspectRatioPixmapLabel::sizeHint() const

{

int w = this->width();

return QSize( w, heightForWidth(w) );

}

QPixmap AspectRatioPixmapLabel::scaledPixmap() const

{

return pix.scaled(this->size(), Qt::KeepAspectRatio, Qt::SmoothTransformation);

}

void AspectRatioPixmapLabel::resizeEvent(QResizeEvent \* e)

{

if(!pix.isNull())

QLabel::setPixmap(scaledPixmap());

}

void AspectRatioPixmapLabel::setClicable(bool x)

{

is\_clicable = x;

}

void AspectRatioPixmapLabel::clearImage()

{

QLabel::clear();

QLabel::setText("NONE");

pix = QPixmap();

}

QString AspectRatioPixmapLabel::getPath\_to\_IMG()

{

return path\_to\_IMG;

}

void AspectRatioPixmapLabel::setCurrentIMG(QString path\_to\_IMG)

{

QPixmap pix(path\_to\_IMG);

this->path\_to\_IMG = path\_to\_IMG;

this->setPixmap(pix);

}

void AspectRatioPixmapLabel::dragEnterEvent(QDragEnterEvent \*event)

{

if(event->mimeData()->hasUrls() && is\_clicable)

event->acceptProposedAction();

}

void AspectRatioPixmapLabel::dropEvent(QDropEvent \*event)

{

const QMimeData\* mimeData = event->mimeData();

if(mimeData->hasUrls() && is\_clicable)

{

QList<QUrl> urlList = mimeData->urls();

if(!urlList.isEmpty())

{

QUrl url = urlList.first();

QString filePath = url.toLocalFile();

QPixmap pix(filePath);

this->setPixmap(pix);

path\_to\_IMG = filePath;

is\_changed = true;

}

}

}

void AspectRatioPixmapLabel::mousePressEvent(QMouseEvent \*e)

{

if(e->button() == Qt::RightButton && is\_clicable)

{

QLabel::clear();

QLabel::setText("NONE");

pix = QPixmap();

path\_to\_IMG = "NONE";

is\_changed = true;

}

}

//dialogforcreatingform.cpp

#include "dialogforcreatingform.h"

DialogForCreatingForm::DialogForCreatingForm( QWidget\* parent ) : QDialog( parent ) {

QBoxLayout\* layout = new QHBoxLayout;

m\_edit = new QLineEdit;

layout->addWidget( m\_edit );

QPushButton\* okBtn = new QPushButton( "OK" );

connect( okBtn, SIGNAL( clicked() ), SLOT( accept() ) );

layout->addWidget( okBtn );

QPushButton\* cancelBtn = new QPushButton( "Cancel" );

connect( cancelBtn, SIGNAL( clicked() ), SLOT( reject() ) );

layout->addWidget( cancelBtn );

setLayout( layout );

}

DialogForCreatingForm::~DialogForCreatingForm() {

}

QString DialogForCreatingForm::getInput() const {

return m\_edit->text();

}

//qrightclickbutton.cpp

#include "qrightclickbutton.h"

QRightClickButton::QRightClickButton(QWidget \*parent) :

QPushButton(parent)

{

}

void QRightClickButton::mousePressEvent(QMouseEvent \*e)

{

if(e->button()==Qt::RightButton && QApplication::keyboardModifiers() == Qt::ShiftModifier)

emit shiftRightClicked();

if(e->button()==Qt::LeftButton)

emit leftClicked();

}

void QRightClickButton::mouseDoubleClickEvent(QMouseEvent \*e)

{

if(e->button() == Qt::RightButton && e->button() == Qt::RightButton)

{

emit doubleRightClicked();

}else{

QWidget::mouseDoubleClickEvent(e);

}

}

//readqfile.h

#include "readqfile.h"

//parses all "Test %1.txt".arg in QMap<int, QMap<QString, QString> and

// QMap<String, String> has folowing keys ( in switch construction u can check

// In wrapper map int - is number of slide

QMap<int, QMap<QString, QString>> parse\_txt\_to\_test (const QString& path)

{

QMap<int, QMap<QString, QString>> result;

QMap<QString, QString> test\_slide;

QString word = "";

QFile file(path);

if(!file.open(QIODevice::OpenModeFlag::ReadOnly))

QApplication::exit();

QTextStream in(&file);

int counter\_first = 0;

while(!in.atEnd())

{

int counter = 0;

QString line = in.readLine();

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

{

if(line[i] == '|')

{

switch(counter)

{

case 0:

test\_slide["img"] = word;

word.clear();

break;

case 1:

test\_slide["question"] = word;

word.clear();

break;

case 2:

test\_slide["amount\_of\_questions"] = word;

word.clear();

break;

case 3:

test\_slide["number\_of\_right\_answer"] = word;

word.clear();

break;

case 4:

test\_slide["variants\_of\_answers"] = word;

word.clear();

break;

}

counter++;

continue;

}

word.append(line[i]);

}

result[counter\_first] = test\_slide;

counter\_first++;

}

return result;

}

QString read\_needed\_line\_from\_file(const QString& path, int num\_of\_line)

{

QString word = "";

QFile file(path);

if(!file.open(QIODevice::OpenModeFlag::ReadOnly))

QApplication::exit();

QTextStream in(&file);

int counter = 0;

while(!in.atEnd())

{

if(counter == num\_of\_line)

word = in.readLine();

}

return word;

}

//parse line into (espessially in "variants\_of\_answers: ) strings and put it in QStringList

QStringList parse\_line\_into\_questions(const QString &line)

{

QStringList result;

QString word;

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

{

if( (line[i] == '/'))

{

result.append(word);

word.clear();

continue;

}

word.append(line[i]);

}

result.append(word);

return result;

}

//writes QMap<int, QMap<QString, QString>> back in the file

void put\_map\_into\_file(const QMap<int, QMap<QString, QString>>& map, QString path\_to\_txt\_file)

{

QFile file(path\_to\_txt\_file);

if(!file.open(QIODevice::OpenModeFlag::WriteOnly))

QApplication::exit();

QTextStream in(&file);

for(QMap<QString, QString> i : map)

{

in << i["img"] << "|" << i["question"] << "|" << i["amount\_of\_questions"] << "|" << i["number\_of\_right\_answer"]

<< "|" << i["variants\_of\_answers"] << "|\n";

}

}

//prints all test with all slides in qDebug

void print\_test\_in\_debug(){

QString path = "alltests/Test1/Test1.txt";

QMap<int,QMap<QString, QString>> result = parse\_txt\_to\_test(path);

for(auto i = result.cbegin(), end = result.cend(); i != end; ++i)

{

for(auto j = i.value().cbegin(), end\_j = i.value().cend(); j != end\_j; ++j )

{

if(j.key() == "variants\_of\_answers")

{

qDebug() << j.key() << '\n';

QStringList tmp = parse\_line\_into\_questions(j.value());

for(QString k : tmp)

{

qDebug() << k << " ";

}

qDebug() << '\n';

continue;

}

qDebug() << j.key() << " " << j.value() << '\n';

}

qDebug() << '\n';

}

}

//checks, if it is avaiable to write QMap<int, QMap<QString, QString> > in file

bool is\_avaiable\_to\_write\_in\_txt\_file(const QMap<int, QMap<QString, QString> > &map)

{

for(QMap<QString, QString> i : map)

{

if(i["amount\_of\_questions"] == "NONE" || i["number\_of\_right\_answer"] == "NONE")

return false;

}

return true;

}

QStringList count\_files\_in\_dir(QString path)

{

QDir directory(path);

QStringList list\_of\_directories = directory.entryList(QDir::NoDotAndDotDot | QDir::AllEntries);

return list\_of\_directories;

}

//aspectratiopixmaplabel\_h

#ifndef ASPECTRATIOPIXMAPLABEL\_H

#define ASPECTRATIOPIXMAPLABEL\_H

#include <QLabel>

#include <QPixmap>

#include <QResizeEvent>

#include <QDragEnterEvent>

#include <QMimeData>

#include <QDropEvent>

#include <QList>

#include <QUrl>

#include <QMouseEvent>

class AspectRatioPixmapLabel : public QLabel

{

Q\_OBJECT

public:

explicit AspectRatioPixmapLabel(QWidget \*parent = 0);

virtual int heightForWidth( int width ) const;

virtual QSize sizeHint() const;

QPixmap scaledPixmap() const;

bool is\_changed = false;

public slots:

void setPixmap ( const QPixmap & );

void resizeEvent(QResizeEvent \*);

void setClicable(bool x);

void clearImage();

QString getPath\_to\_IMG();

void setCurrentIMG(QString path\_to\_IMG);

private slots:

void dragEnterEvent(QDragEnterEvent \*event);

void dropEvent(QDropEvent \*event);

void mousePressEvent(QMouseEvent\* e);

private:

bool is\_clicable = false;

QPixmap pix;

QString path\_to\_IMG = "NONE";

};

#endif // ASPECTRATIOPIXMAPLABEL\_H

//creatingtestform.h

#ifndef CREATINGTESTFORM\_H

#define CREATINGTESTFORM\_H

#include <QWidget>

#include <QString>

#include <QDir>

#include <QFile>

#include <QMap>

#include <QLineEdit>

#include <QList>

#include <QMessageBox>

#include <QPixmap>

#include <QHBoxLayout>

#include <QFileInfo>

#include <QKeyEvent>

#include "aspectratiopixmaplabel.h"

#include "dialogforcreatingform.h"

#include "readqfile.h"

#include "qrightclickbutton.h"

namespace Ui {

class CreatingTestForm;

}

class CreatingTestForm : public QWidget

{ Q\_OBJECT

public:

explicit CreatingTestForm(QWidget \*parent = nullptr);

void set\_path\_to\_test(const QString& path);

void set\_name\_of\_test(const QString& name);

~CreatingTestForm();

public slots:

void generate\_creating\_form();

void setCreatingFormForEditSlide();

private slots:

void cancelPushButton\_clicked();

void on\_editQuestionsPushButton\_clicked();

void on\_cancelPushButton\_2\_clicked();

void on\_addQuestionPushButton\_clicked();

void onRemovedButton();

void on\_savePushButton\_2\_clicked();

void chooseRightAnswer();

void on\_nextSlideButton\_clicked();

void on\_prevPushButton\_clicked();

void on\_savePushButton\_clicked();

void on\_addImagePushButton\_clicked();

void generate\_image();

void returnPushButton\_clicked();

bool is\_need\_to\_delete\_IMG(const QString& path\_to\_img);

void keyPressEvent(QKeyEvent\* e);

void removeCurrentPage(int index\_to\_remove);

private:

void create\_current\_info\_map(QMap<int, QMap<QString, QString>>& tmp\_map,int num\_of\_slide);

bool is\_avaiable\_nextSlidePushButton();

QString get\_number\_of\_right\_answer(const QList<QRightClickButton\*>& list);

signals:

void open\_menuwindow();

private:

QString num\_of\_right\_answer = "NONE";

QString info\_in\_text\_file = "%1|%2|%3|%4|%5";

Ui::CreatingTestForm \*ui;

QList<QRightClickButton\*> buttons\_list;

AspectRatioPixmapLabel\* imgLabel;

QMap<int, QMap<QString, QString>> myTest;

QString path\_to\_test;

QString name\_of\_test;

QString path\_to\_IMG;

bool is\_chosen = false;

bool is\_next = false;

bool is\_editor = false;

int num\_of\_slide = 0;

};

#endif // CREATINGTESTFORM\_H

//dialogforcreatingform.h

#ifndef DIALOGFORCREATINGFORM\_H

#define DIALOGFORCREATINGFORM\_H

#include <QDialog>

#include <QDir>

#include <QString>

#include <QLayout>

#include <QLineEdit>

#include <QPushButton>

class DialogForCreatingForm : public QDialog {

Q\_OBJECT

public:

DialogForCreatingForm( QWidget\* parent = 0 );

~DialogForCreatingForm();

QString getInput() const;

signals:

void open\_creating\_tests\_page();

private:

QLineEdit\* m\_edit;

};

#endif // DIALOGFORCREATINGFORM\_H

//horizontalscrollarea.h

#ifndef HORIZONTALSCROLLAREA\_H

#define HORIZONTALSCROLLAREA\_H

#include <QGridLayout>

#include <QResizeEvent>

#include <QScrollArea>

#include <QScrollBar>

class HorizontalScrollArea : public QScrollArea

{

QWidget \*contentWidget;

QGridLayout \*grid;

int nRows;

int nColumns;

public:

HorizontalScrollArea(int rows, int cols, QWidget \*parent = Q\_NULLPTR)

:QScrollArea(parent), nRows(rows), nColumns(cols)

{

setWidgetResizable(true);

contentWidget = new QWidget(this);

setWidget(contentWidget);

grid = new QGridLayout(contentWidget);

grid->setHorizontalSpacing(5);

setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

}

void addWidget(QWidget \*w, int row, int col){

grid->addWidget(w, row, col);

adaptSize();

}

int columnCount() const{

if(grid->count() == 0){

return 0;

}

return grid->columnCount();

}

private:

void adaptSize(){

if(columnCount() >= nColumns ){

int w = 1.0\*(width() - grid->horizontalSpacing()\*(nColumns+1.6))/nColumns;

int wCorrected = w\*columnCount() + grid->horizontalSpacing()\*(columnCount()+2);

contentWidget->setFixedWidth(wCorrected);

}

contentWidget->setFixedHeight(viewport()->height());

}

protected:

void resizeEvent(QResizeEvent \*event){

QScrollArea::resizeEvent(event);

adaptSize();

}

};

#endif // HORIZONTALSCROLLAREA\_H

#ifndef MENUWINDOW\_H

#define MENUWINDOW\_H

#include <QDebug>

#include <QLabel>

#include <QTime>

//menuwindow.h

#include <QMainWindow>

#include <QDir>

#include <QString>

#include <QGridLayout>

#include <QSplitter>

#include <QPushButton>

#include <QFile>

#include <QMouseEvent>

#include "readqfile.h"

#include "testwidget.h"

#include "horizontalscrollarea.h"

#include "testwidget.h"

#include "creatingtestform.h"

#include "qrightclickbutton.h"

namespace Ui {

class MenuWindow;

}

class MenuWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MenuWindow(QWidget \*parent = nullptr);

~MenuWindow();

private slots:

void dynamicPushButton\_clicked();

void on\_testsPushButton\_clicked();

void creating\_Tests\_page();

void on\_createPushButton\_clicked();

void shift\_right\_dynamicPushButton\_clicked();

void double\_right\_dynamicPushButton\_clicked();

void on\_exitPushButton\_clicked();

signals:

void open\_creating\_tests\_form();

void open\_menuwindow();

private:

Ui::MenuWindow \*ui;

HorizontalScrollArea\* scroll;

TestWidget\* testWidget;

CreatingTestForm\* testForm;

QVector<QRightClickButton\*> dynamic\_buttons;

};

#endif // MENUWINDOW\_H

//qrightclickbutton.h

#ifndef QRIGHTCLICKBUTTON\_H

#define QRIGHTCLICKBUTTON\_H

#include <QPushButton>

#include <QMouseEvent>

#include <QKeyEvent>

#include <QApplication>

class QRightClickButton : public QPushButton

{

Q\_OBJECT

public:

explicit QRightClickButton(QWidget \*parent = 0);

private slots:

void mousePressEvent(QMouseEvent \*e);

void mouseDoubleClickEvent(QMouseEvent\* e);

signals:

void shiftRightClicked();

void leftClicked();

void doubleRightClicked();

};

#endif // QRIGHTCLICKBUTTON\_H

//readqfile.h

#ifndef READQFILE\_H

#define READQFILE\_H

#include <QString>

#include <QMap>

#include <QFile>

#include <QApplication>

#include <QDebug>

#include <QDir>

QMap<int, QMap<QString, QString>> parse\_txt\_to\_test (const QString& path);

void print\_test\_in\_debug();

QStringList parse\_line\_into\_questions(const QString& line);

void put\_map\_into\_file(const QMap<int, QMap<QString, QString>>& map, QString path\_to\_txt\_file);

bool is\_avaiable\_to\_write\_in\_txt\_file(const QMap<int, QMap<QString, QString>>& map);

QStringList count\_files\_in\_dir(QString path);

#endif // READQFILE\_H

//testwidget.h

#ifndef TESTWIDGET\_H

#define TESTWIDGET\_H

#include <QWidget>

#include <QLabel>

#include <QPushButton>

#include <QMessageBox>

#include <QPixmap>

#include <QKeyEvent>

#include <QKeyEvent>

#include "aspectratiopixmaplabel.h"

#include "horizontalscrollarea.h"

#include "readqfile.h"

namespace Ui {

class TestWidget;

}

class TestWidget : public QWidget

{

Q\_OBJECT

public:

explicit TestWidget(QWidget \*parent = nullptr);

~TestWidget();

void add\_answers();

private slots:

void action\_if\_right\_answer();

void delete\_dynamic\_buttons();

void generate\_slide();

void add\_image\_on\_slide();

void change\_style\_if\_right\_answer();

void make\_buttons\_unclickable();

public:

void setPath();

signals:

void activate\_generating\_slide();

void open\_menuwindow();

void continue\_test();

protected:

void keyPressEvent(QKeyEvent \*e) override;

private:

Ui::TestWidget \*ui;

QMap<int, QMap<QString, QString>> myTest;

QVector<QPushButton\*> button\_vector;

QString path\_to\_test;

HorizontalScrollArea\* scroll;

AspectRatioPixmapLabel\* imgLabel;

bool is\_image = false;

bool can\_we\_continue = false;

int num\_of\_slide = 0;

int score = 0;

};

#endif // TESTWIDGET\_H