ЛАБОРАТОРНАЯ РАБОТА №7

«ИССЛЕДОВАНИЕ СПОСОБОВ РЕАЛИЗАЦИИ СЕТЕВЫХ ЗАПРОСОВ В QT-ПРИЛОЖЕНИЯХ»

1. Цель работы

Исследование способов работы Qt-приложения с HTTP GET/POST запросами. Приобретение навыков разработки простейших сетевых приложений.

2. Постановка задачи

2.1. Изучить основные возможности модуля QtNetwork для реализации HTTP-запросов (выполняется в ходе самостоятельной подготовки к лабораторной работе).

2.2. Создать Qt GUI приложение

2.3. В дизайнере создать форму со следующим расположением элементов управления (рисунок 1).

2.4. Реализовать следующую логику приложения. Все элементы, кроме кнопок выбора типа запроса изначально скрыты. После выбора метода отобразить скрытые элементы. При этому кнопки выбора метода – скрыть. QLabel с текстом “Enter Data:” и соответствующее поле ввода должно отображаться только для POST запроса. По нажатию кнопки Submit реализовать отправку соответствующего запроса и вывод результата в поле Response. Для POST запроса использовать данные из поля Enter Data.

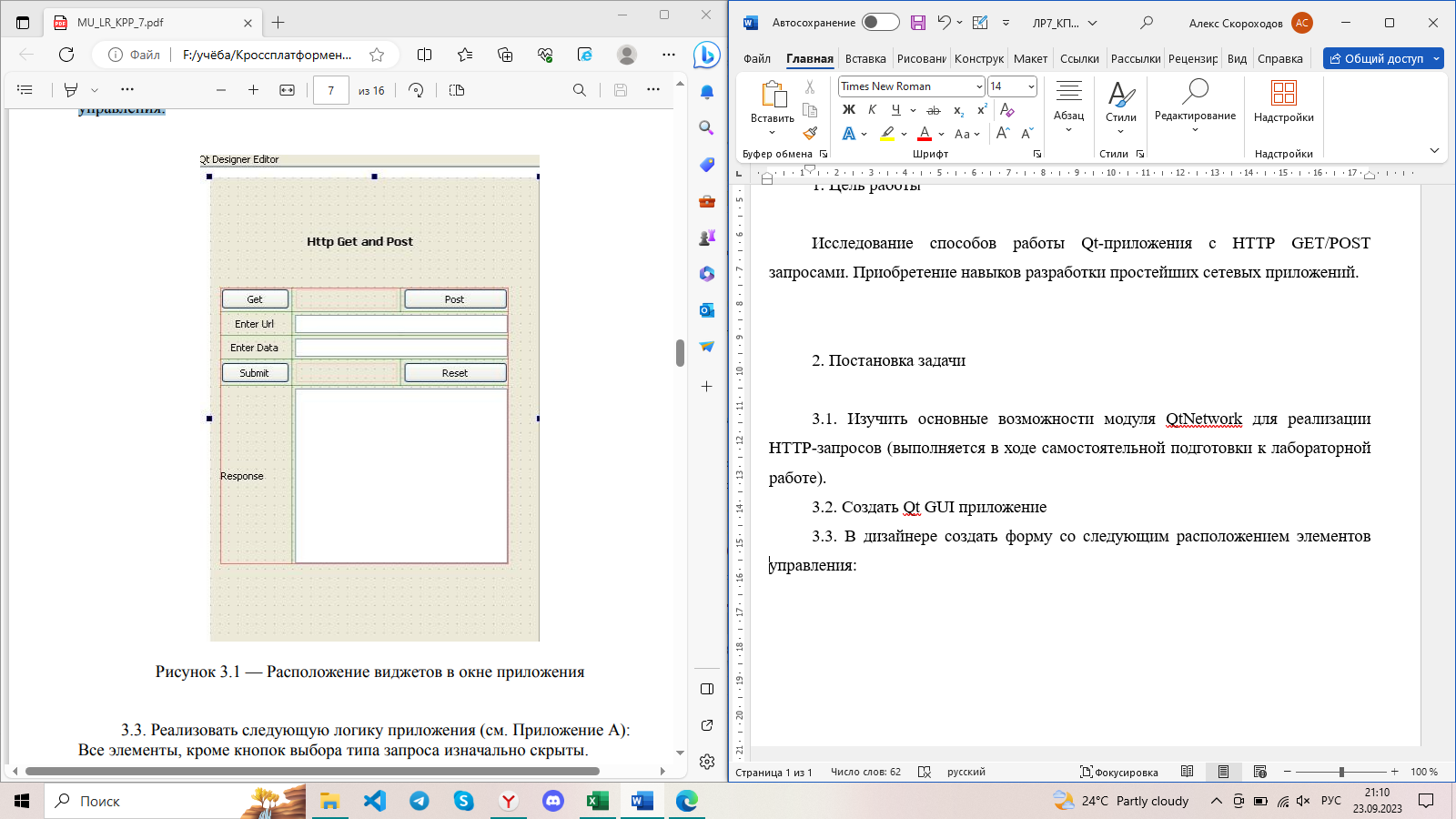


Рисунок 1 – Форма для GET- и POST-запросов

3. Ход работы

Был написан код, представленный в листингах 1-3, который реализует функциональность по заданию.

Листинг 1 – Код файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QMessageBox>

#include<QByteArray>

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow {

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = 0);

~***MainWindow***();

protected:

void ***changeEvent***(QEvent \*e);

private slots:

void **finished**(QNetworkReply \*reply);

void **DoHttpGet**();

void **activateGetWidgets**();

void **HideWidgets**();

void **clearWidgets**();

void **activatePostWidgets**();

private:

Ui::MainWindow \*ui;

QNetworkAccessManager \*nam;

};

#endif // MAINWINDOW\_H

Листинг 2 – Код файла main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

Листинг 3 – Код файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

nam = new QNetworkAccessManager(this);

HideWidgets();

connect(ui->getButton,SIGNAL(clicked()),this,SLOT(activateGetWidgets()));

connect(ui->submitButton,SIGNAL(clicked()),this,SLOT(DoHttpGet()));

connect(ui->resetButton,SIGNAL(clicked()),this,SLOT(clearWidgets()));

connect(nam,SIGNAL(finished(QNetworkReply\*)),this,SLOT(finished(QNetworkReply\*)));

connect(ui->postButton,SIGNAL(clicked()),this,SLOT(activatePostWidgets()));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::***changeEvent***(QEvent \*e)

{

QMainWindow::changeEvent(e);

switch (e->type()) {

case QEvent::LanguageChange:

ui->retranslateUi(this);

break;

default:

break;

}

}

void MainWindow::**activateGetWidgets**()

{

ui->urlLabel->setHidden(false);

ui->urlLine->setHidden(false);

ui->submitButton->setHidden(false);

ui->textBrowser->setHidden(false);

ui->responseTitleLabel->setHidden(false);

ui->getButton->setHidden(true);

ui->postButton->setHidden(true);

}

void MainWindow::**activatePostWidgets**()

{

ui->dataLabel->setHidden(false);

ui->dataLine->setHidden(false);

activateGetWidgets();

}

void MainWindow::**finished**(QNetworkReply \*reply)

{

if(reply->error() == QNetworkReply::NoError)

{

ui->textBrowser->setText(reply->readAll());

}

else

{

ui->textBrowser->setText(reply->errorString());

}

}

void MainWindow::**DoHttpGet**()

{

ui->resetButton->setHidden(false);

QString url = ui->urlLine->text();

QString data = ui->dataLine->text();

QByteArray postData;

postData.append(data.toLatin1());

if(postData.isEmpty() == true)

{

nam->get(QNetworkRequest(QUrl(url)));

}

else

{

nam->post(QNetworkRequest(QUrl(url)),postData);

}

}

void MainWindow::**HideWidgets**()

{

ui->urlLabel->setHidden(true);

ui->urlLine->setHidden(true);

ui->dataLabel->setHidden(true);

ui->dataLine->setHidden(true);

ui->submitButton->setHidden(true);

ui->responseTitleLabel->setHidden(true);

ui->textBrowser->setHidden(true);

ui->resetButton->setHidden(true);

}

void MainWindow::**clearWidgets**()

{

ui->urlLabel->setHidden(true);

ui->urlLine->setHidden(true);

ui->dataLabel->setHidden(true);

ui->dataLine->setHidden(true);

ui->submitButton->setHidden(true);

ui->responseTitleLabel->setHidden(true);

ui->textBrowser->setHidden(true);

ui->resetButton->setHidden(true);

ui->urlLine->clear();

ui->textBrowser->clear();

ui->dataLine->clear();

ui->getButton->setHidden(false);

ui->postButton->setHidden(false);

}

Также была создана форма с необходимыми кнопками, вид формы представлен на рисунках 2-4.

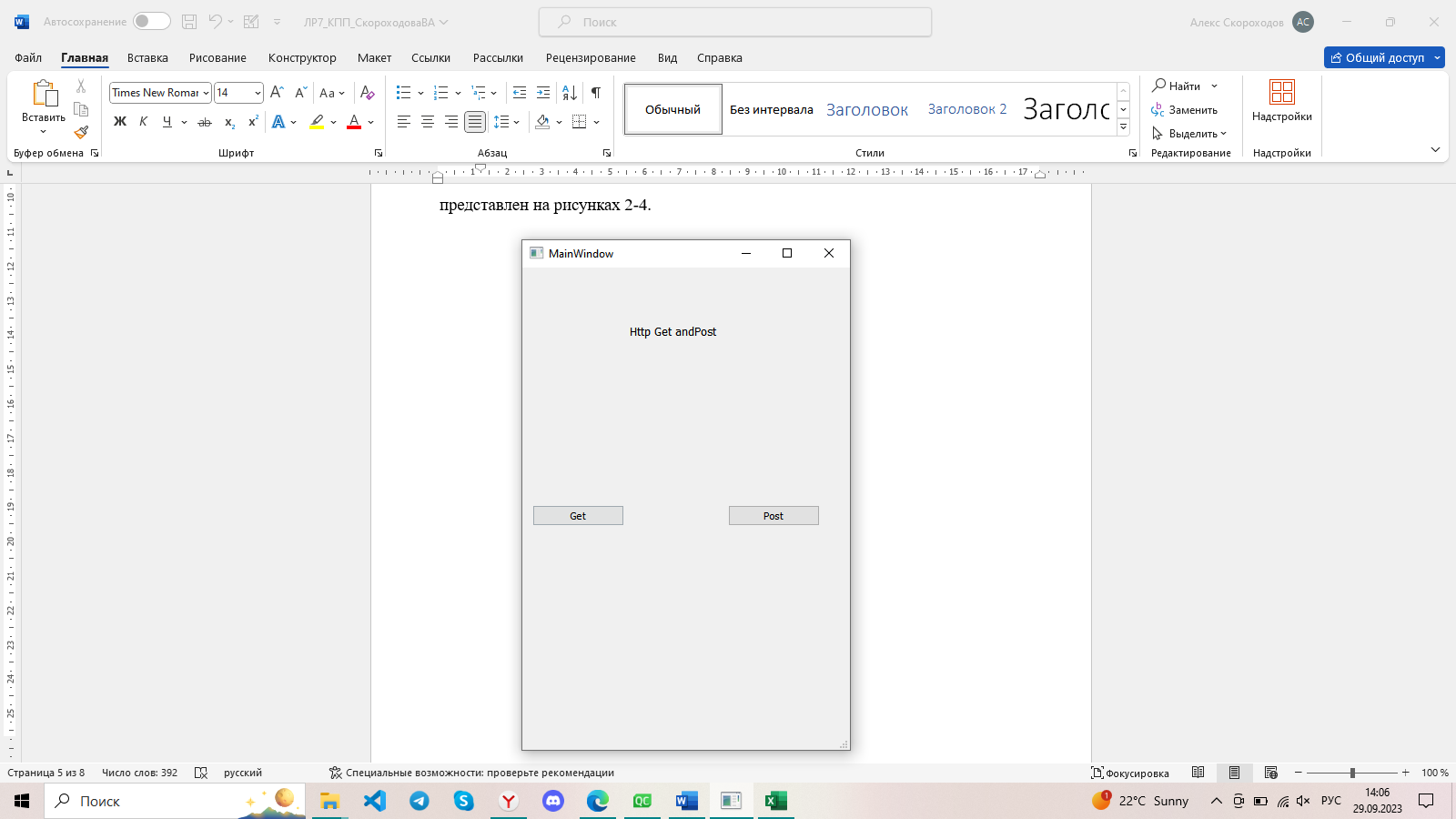


Рисунок 2 – Вид приложения в самом начале

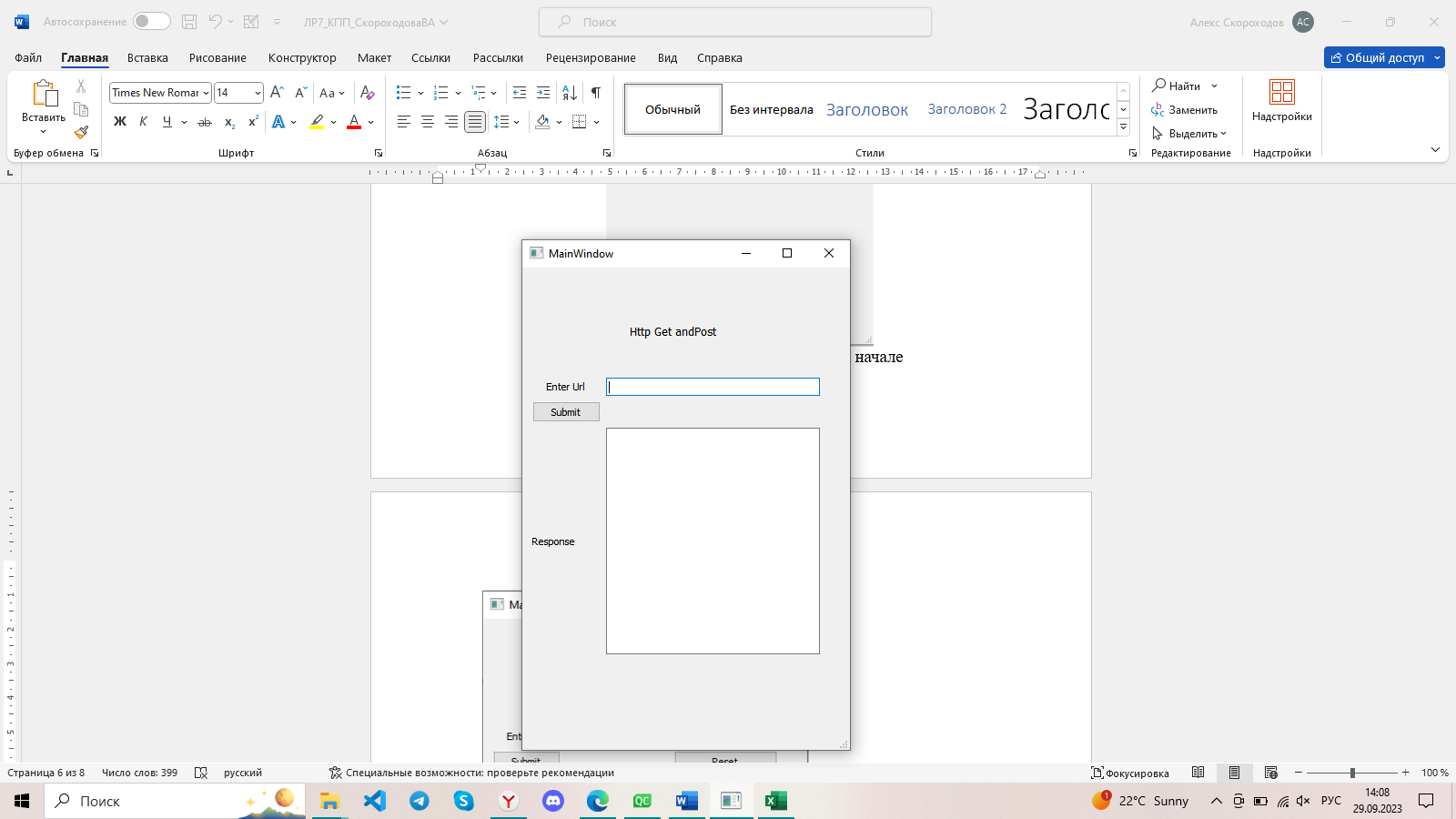


Рисунок 3 – Раздел для отправки GET-запросов

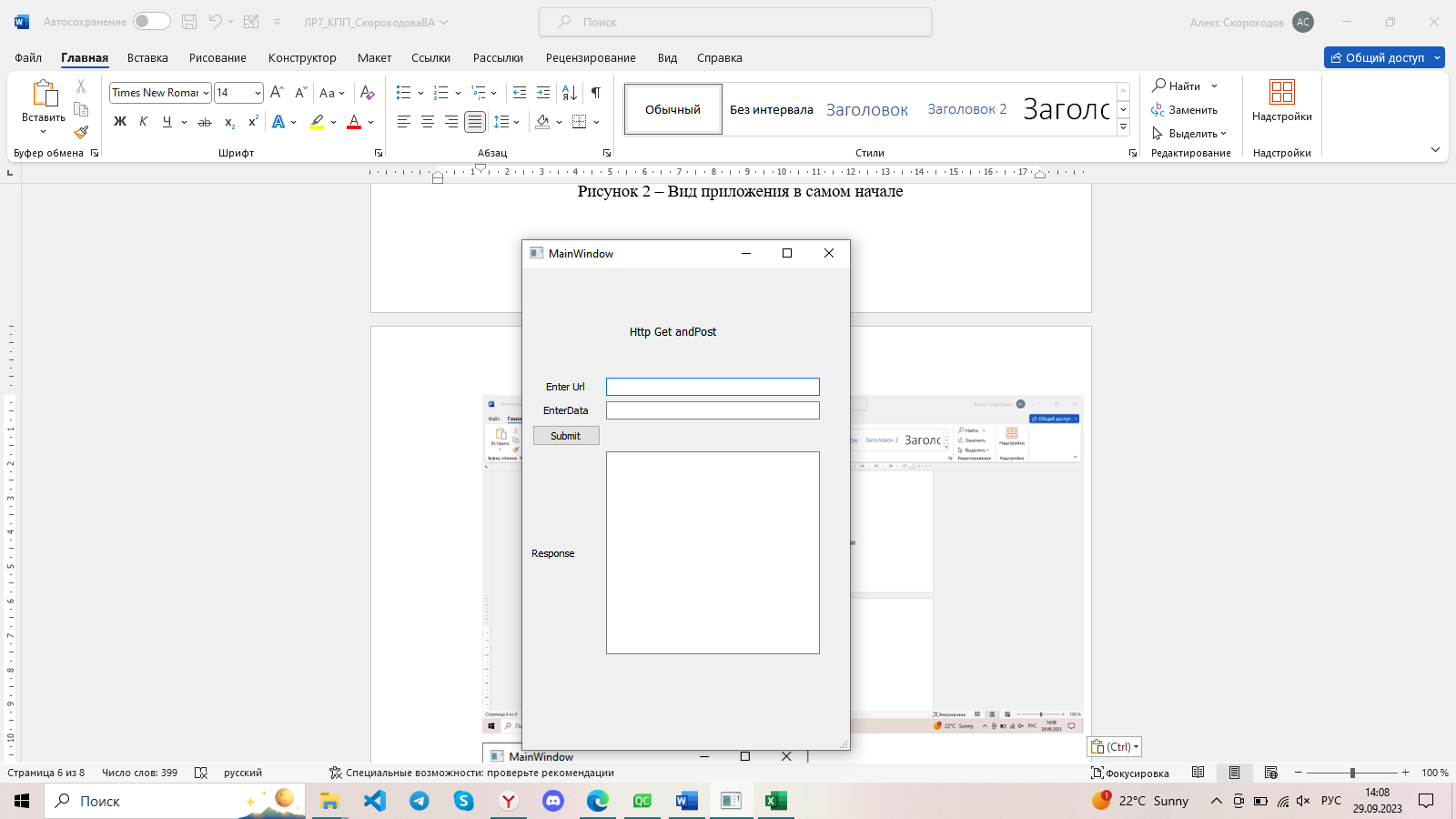


Рисунок 4 – Раздел для отправки POST-запросов

Для тестирования работы GET-запроса был введен следующий URL: <http://forismatic.com/ru/api/>

Результат работы запроса представлен на рисунке 5. Как видно из теста, GET-запрос вернул содержимое HTML-страницы.

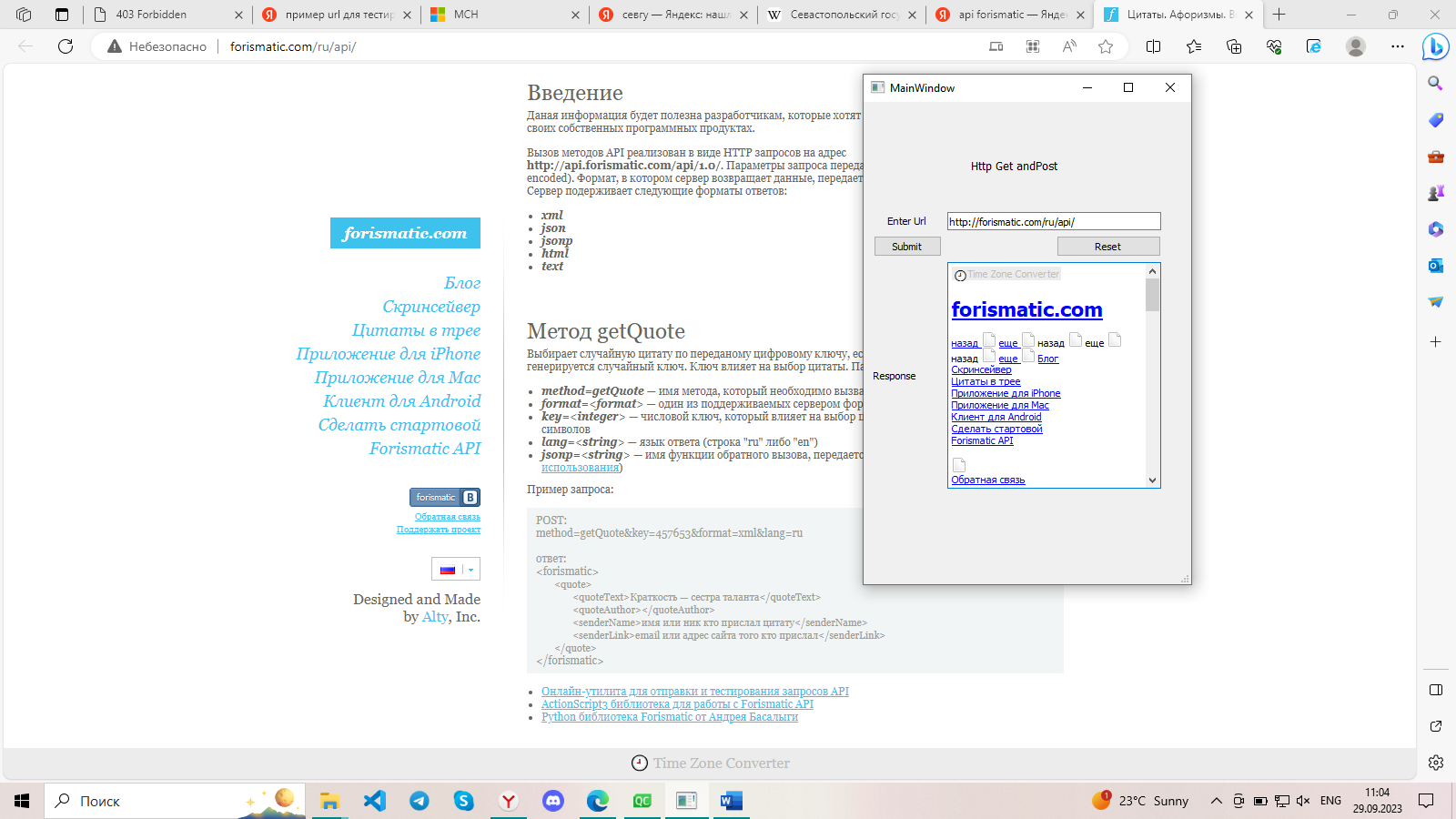


Рисунок 5 – Результат работы GET-запроса

Далее была протестирована работа POST-запросов. На рисунке 6 изображен результат работы.

Текст запроса:

URL: <http://api.forismatic.com/api/1.0/>

Методы: method=getQuote&key=457653&format=xml&lang=ru

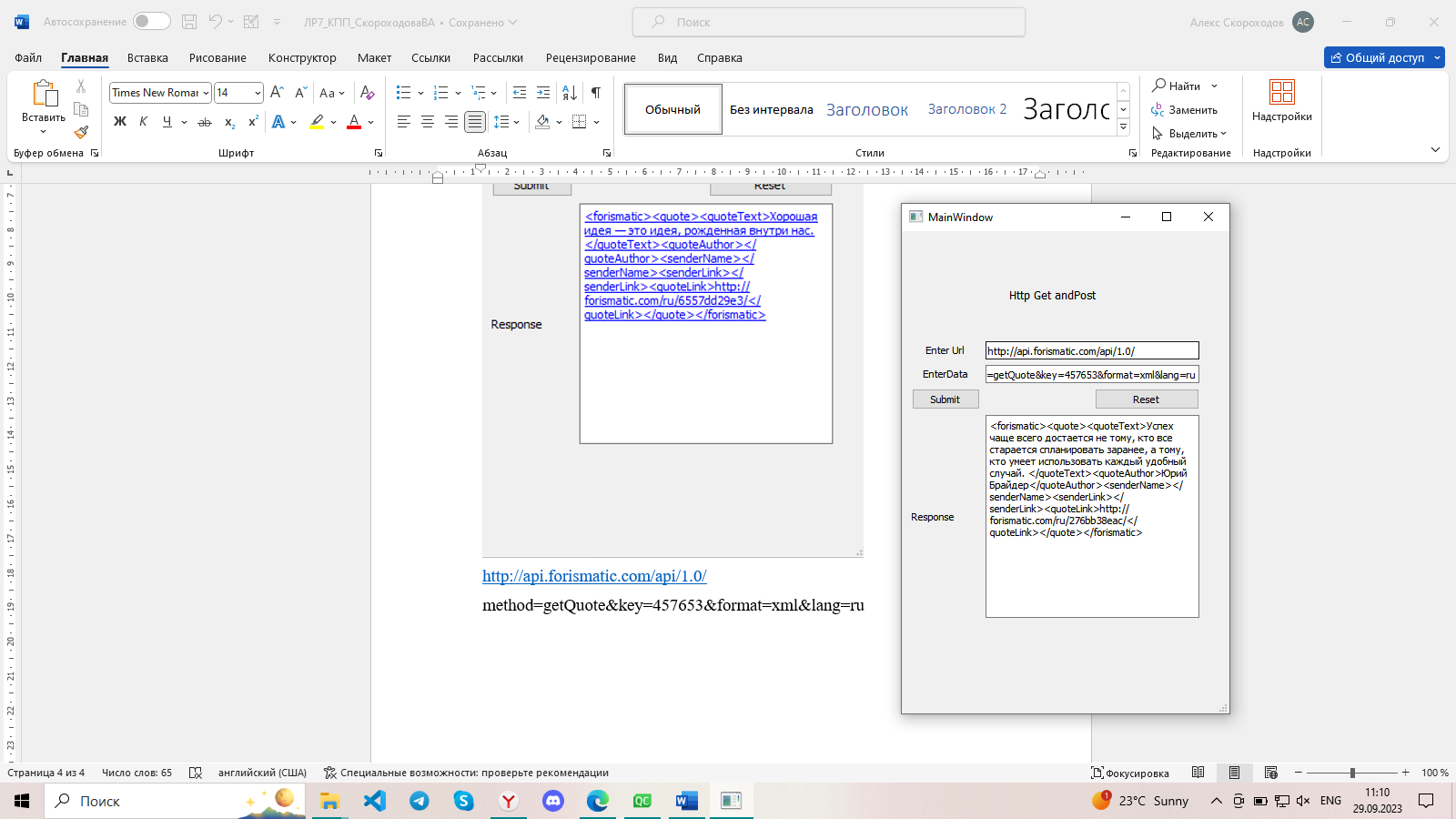


Рисунок 6 – Результат работы POST-запроса

ВЫВОДЫ

В ходе лабораторной работы было проведено исследование способов работы Qt-приложения с HTTP GET/POST запросами. Было написано приложение, которое обрабатывало GET- и POST-запросы, в следствие чего были приобретены навыки разработки простейших сетевых приложений. Было проведено тестирование приложения, и по результатам можно судить, что приложение работает корректно.

Контрольные вопросы:

5.1. Какой модуль Qt Framework предназначен для работы с TCP/IP протоколом?

**QtNetwork**

5.2. Какие основные классы используются для работы с сетевыми операциями?

Он предлагает классы, такие как QHttp и QFtp, которые реализуют специфические протоколы уровня приложения (application-level), низкоуровневые классы, такие как QTcpSocket, QTcpServer и QUdpSocket, которые представляют низкоуровневые сетевые концепции, и высокоуровневые классы, такие как QNetworkRequest, QNetworkReply и QNetworkAccessManager для представления сетевых операций с использованием распространенных протоколов.

5.3. Расскажите о назначении класса QNetworkAccessManager.

Класс QNetworkAccessManager позволяет приложению отправлять сетевые запросы и получать ответы.

API сетевого доступа создано вокруг объекта QNetworkAccessManager, который содержит общую конфигурацию и настройки для посылаемых запросов. Он содержит прокси и кэш, а также сигналы, связанные с ними, и сигналы ответов, которые могут быть использованы для контроля за прогрессом сетевой операции. Одного объекта QNetworkAccessManager будет достаточно для всего приложения Qt.

После создания объекта QNetworkAccessManager, приложение может посылать запросы по сети. Поставляется группа стандартных функций, которые принимают запрос и необязательные данные, и каждая возвращает объект QNetworkReply. Возвращаемый объект используется для получения любых данных, возвращаемых в ответ на соответствующий запрос.

5.4. Каким образом осуществить отправку и получение результата GET запроса, используя QNetworkAccessManager?

5.4. Чтобы осуществить отправку и получение результата GET-запроса с использованием QNetworkAccessManager в Qt, вы можете выполнить следующие действия:

QNetworkAccessManager\* manager = new QNetworkAccessManager(this);

// Создание объекта QNetworkRequest с указанием URL-адреса для GET-запроса

QNetworkRequest request(QUrl("http://example.com/api"));

// Отправка GET-запроса

QNetworkReply\* reply = manager->get(request);

// Обработка ответа

connect(reply, &QNetworkReply::finished, [=]() {

if (reply->error() == QNetworkReply::NoError) {

// Получение данных ответа

QByteArray data = reply->readAll();

// Обработка данных

} else {

// Обработка ошибки

qDebug() << "Error:" << reply->errorString();

}

// Освобождение ресурсов

reply->deleteLater();

});

5.5. Каким образом осуществить отправку и получение результата POST запроса, используя QNetworkAccessManager?

5.5. Чтобы осуществить отправку и получение результата POST-запроса с использованием QNetworkAccessManager в Qt, вы можете выполнить следующие действия:

QNetworkAccessManager\* manager = new QNetworkAccessManager(this);

// Создание объекта QNetworkRequest с указанием URL-адреса для POST-запроса

QNetworkRequest request(QUrl("http://example.com/api"));

// Установка заголовка Content-Type для указания типа данных запроса

request.setHeader(QNetworkRequest::ContentTypeHeader, "application/x-www-form-urlencoded");

// Создание тела запроса в формате URL-кодирования

QByteArray postData;

postData.append("param1=value1&param2=value2");

// Отправка POST-запроса

QNetworkReply\* reply = manager->post(request, postData);

// Обработка ответа

connect(reply, &QNetworkReply::finished, [=]() {

if (reply->error() == QNetworkReply::NoError) {

// Получение данных ответа

QByteArray data = reply->readAll();

// Обработка данных

} else {

// Обработка ошибки

qDebug() << "Error:" << reply->errorString();

}

// Освобождение ресурсов

reply->deleteLater();

});

5.6. Какую роль играет механизм сигналов/cлотов при отправке и обработке результата запроса?

5.6. Механизм сигналов/слотов в Qt играет важную роль при отправке и обработке результата запроса с использованием QNetworkAccessManager. Он позволяет связывать сигналы, генерируемые QNetworkAccessManager и QNetworkReply, с соответствующими слотами для обработки событий. Например, сигнал finished объекта QNetworkReply сигнализирует о завершении запроса, и вы можете подключить его к слоту для обработки ответа или ошибки.

5.7. Необходимо ли для осуществления отдельного запроса использовать отдельный экземпляр QNetworkAccessManager? Если нет, каким образом вы бы реализовали работу с QNetworkAccessManager в рамках крупного проекта? Какие шаблоны проектирования позволили бы упростить эту задачу?

5.7. Для осуществления отдельного запроса не обязательно использовать отдельный экземпляр QNetworkAccessManager. Один экземпляр QNetworkAccessManager может обрабатывать несколько запросов параллельно. В крупном проекте вы можете создать глобальный экземпляр QNetworkAccessManager, доступный из различных частей приложения.

Относительно шаблонов проектирования, паттерн Singleton может быть полезным для создания глобального экземпляра QNetworkAccessManager. Паттерн Observer может быть использован для уведомления различных компонентов приложения о событиях, связанных с запросами и ответами. Также можно применить паттерн Фасад для создания удобного интерфейса для работы с QNetworkAccessManager и скрытия деталей реализации от других компонентов приложения.