

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 7

з дисципліни “ Основи програмування ”

Тема “**TCP клієнт та публічні Web API**”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-62  Лук’янець Михайло Олександрович  (*прізвище, ім’я, по батькові*)  варіант №15 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 2017 р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Штрафні бали:   |  |  | | --- | --- | | **Термін здачі** | **Оформлення звіту** | |  |  | | Нараховані бали:   |  |  |  | | --- | --- | --- | | **Корект. програм (2 бала)** | **Відп. на теор. питання (1 бал)** | **Відп. на прогр. питання (2 бала)** | |  |  |  | | Сумарний бал:   |  | | --- | |  | |

Київ 2017

**Мета роботи**

Навчитися користуватись публічними [API (Application Programming Interface)](https://en.wikipedia.org/wiki/Application_programming_interface) web-сайтів у мережі Інтернет за допомогою TCP клієнта та парсерів форматів даних.

**Постановка завдання**

#### **Крок №1. Створення TCP-клієнта**

Замітка: даний крок призначений для ознайомлення зі способом підключення і отримання даних від web-сервера по протоколу HTTPS для використання у кроці №2. Якщо Ви використовуєте libprogbase, встановіть sudo apt-get install libssl-dev та оновіть libprogbase до версії >= 0.3.4. У запитах використовуйте версію протоколу HTTP/1.0.

Реалізувати модуль, що використовує TCP клієнт для підключення до [тестового web-сервера](https://api.github.com/).

Для виконання HTTP(S) GET запиту на отримання даних із web-сервера через TCP потрібно:

1. Виконати підключення TCP клієнта до сервера із хост-іменем api.github.com (потрібно програмно отримати IP-адресу сервера). Порт обирається відповідно до протоколу доступу (80 для HTTP або 443для HTTPS).
2. Після успішного з'єднання сформувати строку HTTP GET запиту виду: GET %s HTTP/1.0\r\nHost: %s\r\n\r\n, де замість першого параметру форматування задати URL-шлях, що розташований за хост-іменем (наприклад, /), а замість другого параметру - хост-ім'я сайту (наприклад, api.github.com).
3. Відправити сформовану строку на сервер і отримувати відповідь доки розмір отриманих у відповідь байт не буде 0. Всі частини відповіді об'єднати у одну строку. Видалити зі строки всі символи, що розміщені перед двома end-of-line підряд (end-of-line: \n (Unix) або \r\n (Windows) . Результат звірити з JSON строкою, що відображається при переході на сайт у браузері.

#### **Крок №2. Отримання та парсинг даних з API**

Дано публічний [API Sunrise-sunset](https://sunrise-sunset.org/api) за URI https://sunrise-sunset.org/api. Виконуючи HTTP(S) GET запити по різних шляхах даного сайту можна отримувати у відповідь дані у строці формату JSON. Використовуючи класи платформи Qt для роботи із JSON розпарсити отриману строку у JSON-дерево. Створити клас SunData. Створити список міст із їх геокординатами. Описати функцію, що за допомогою запропонованого API та списку координат країн дозволятиме отримувати дані про час сходу та заходу сонця.

Зверніть увагу на те, що кількість запитів до API протягом дня може бути обмеженою і після N-нного запиту в день дані стануть недоступні для вас аж до наступного дня.

#### **Крок №3. Відображення даних у віконних програмах**

Створити віконну програму Qt, що дає користувачу можливість обрати одне із заданих міст. Візуалізувати час сходу та заходу сонця у обраному місті а також поточну позицію сонця відносно сходу-заходу. Вивести додаткову інформацію: наприклад, скільки часу пройшло від сходу і скільки залищилось до заходу.

**Тексти коду програм**

|  |
| --- |
| mainwindow.cpp |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "sundata.h"  #include <QString>  #include <QDebug>  #include "adddialog.h"  #include <QPainter>  #include <time.h>  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  SunData Kyiv("Kyiv", 51, 31);  if(Kyiv.GetCity().compare("ERROR") == 0){  qDebug() << "FAILED TO GET CITY DATA";  return;  }  QVariant qVariant;  qVariant.setValue(Kyiv);  QListWidgetItem \*qCityListItem = new QListWidgetItem();  qCityListItem->setText(QString::fromStdString(Kyiv.GetCity()));  qCityListItem->*setData*(Qt::*UserRole*, qVariant);  ui->City\_listWidget->addItem(qCityListItem);  }  MainWindow::~*MainWindow*()  {  delete ui;  }  void MainWindow::on\_City\_listWidget\_itemSelectionChanged()  {  auto items = ui->City\_listWidget->selectedItems();  if (items.size() == 0) {  return;  }  QListWidgetItem \* selectedItem = items[0];  SunData city = selectedItem->*data*(Qt::*UserRole*).value<SunData>();  ui->City\_label->setText(QString::fromStdString(city.GetCity()));  ui->CurTime\_label->setText(QString::fromStdString(city.GetLocalTime()));  ui->Lat\_label->setText(QString::number(city.GetLat()));  ui->Lng\_label->setText(QString::number(city.GetLng()));  ui->Sunrise\_label->setText(QString::fromStdString(city.GetSunrise()));  ui->Sunset\_label->setText(QString::fromStdString(city.GetSunset()));  ui->UTCShift\_label->setText(QString::fromStdString(city.GetUTCShift()));  ui->ZoneId\_label->setText(QString::fromStdString(city.GetTimeZoneId()));  ui->ZoneName\_label->setText(QString::fromStdString(city.GetTimeZoneName()));  //add something for showing circle as sun and turnong off/on sunavalaible lable  int dayLengthSeconds = abs((city.GetSunsetSeconds() + city.GetUTCShiftSeconds())%(24\*60\*60) -  (city.GetSunriseSeconds() + city.GetUTCShiftSeconds()) % (24\*60\*60));  time\_t thisDaySeconds = city.GetTimeSinceMidnightSeconds() -  abs((city.GetSunriseSeconds() + city.GetUTCShiftSeconds())%(24\*60\*60));  qDebug() << "dayLengthSeconds" << dayLengthSeconds << " thisDaySeconds" << thisDaySeconds <<  "time since midnight" << city.GetTimeSinceMidnightSeconds();  if(thisDaySeconds < 0 || thisDaySeconds > dayLengthSeconds){  ui->SunAvalaible\_label->*setVisible*(true);  ui->Sun\_progressBar->*setVisible*(false);  } else {  ui->SunAvalaible\_label->*setVisible*(false);  ui->Sun\_progressBar->*setVisible*(true);  ui->Sun\_progressBar->setValue(thisDaySeconds \* 100 / dayLengthSeconds);  }  }  void MainWindow::on\_Add\_pushButton\_pressed()  {  adddialog dialog;  dialog.setWindowTitle("Enter data");  if (dialog.*exec*()) {  QString city;  float lng;  float lat;  dialog.GetOptions(city, lat, lng);  SunData City(city.toStdString(), lat, lng);  if(City.GetCity().compare("ERROR") == 0){  qDebug() << "FAILED TO GET CITY DATA";  return;  }  QVariant qVariant;  qVariant.setValue(City);  QListWidgetItem \*qCityListItem = new QListWidgetItem();  qCityListItem->setText(QString::fromStdString(City.GetCity()));  qCityListItem->*setData*(Qt::*UserRole*, qVariant);  ui->City\_listWidget->addItem(qCityListItem);  qDebug() << "Accepted!";  } else {  qDebug() << "Rejected!";  }  } |

|  |
| --- |
| SunData.cpp |
| #include "sundata.h"  #include <string>  #include <time.h>  #include <QDebug>  #include <curlpp/cURLpp.hpp>  #include <curlpp/Easy.hpp>  #include <curlpp/Options.hpp>  #include <sstream>  #include <QJsonDocument>  #include <QJsonObject>  #include <QJsonArray>  #include <stdio.h>  using namespace std;  SunData::SunData(){};  SunData::SunData(string city, float latitude, float length)  {  this->city = city;  this->lat = latitude;  this->lng = length;  setup();  }  bool SunData::parseUtcJsonStr(std::string str){  QJsonDocument jdoc = QJsonDocument::fromJson(QString::fromStdString(str).toUtf8());  if (jdoc.isObject()) {  QJsonObject jobj = jdoc.object();  std::string status = jobj["status"].toString().toStdString();  if(status.compare("OK") == 0){  auto utcShift = jobj["rawOffset"].toInt();  this->UTCShiftSeconds = utcShift;  this->timeZoneId = jobj["timeZoneId"].toString().toStdString();  this->timeZoneName = jobj["timeZoneName"].toString().toStdString();  return true;  } else  return false;  } else {  return false;  }  }  bool SunData::parseTimeJsonStr(std::string str){  QJsonDocument jdoc = QJsonDocument::fromJson(QString::fromStdString(str).toUtf8());  if (jdoc.isObject()) {  QJsonObject jobj = jdoc.object();  std::string status = jobj["status"].toString().toStdString();  if(status.compare("OK") == 0){  auto results = jobj["results"].toObject();  auto sunriseRaw = results["sunrise"].toString().toStdString();  auto sunsetRaw = results["sunset"].toString().toStdString();  qDebug() << "Data received time";  this->sunrise = sunriseRaw;  this->sunset = sunsetRaw;  //for seconds setter  int timeHours = 0;  int timeMinutes = 0;  int timeSeconds = 0;  if( sscanf(sunriseRaw.c\_str(), "%d:%d:%d", &timeHours, &timeMinutes, &timeSeconds) < 3){  qDebug() << "parse time sunrise setting error " << QString::fromStdString(str) ;  return false;  }  std::size\_t found = sunriseRaw.find("PM");  if (found!=std::string::npos){  timeHours += 12;  }  this->sunriseSeconds = timeHours \* 60 \* 60 + timeMinutes \* 60 + timeSeconds;  if(sscanf(sunsetRaw.c\_str(), "%d:%d:%d", &timeHours, &timeMinutes, &timeSeconds) < 3){  qDebug() << "parse time sunset setting error";  return false;  }  found = sunsetRaw.find("PM");  if (found!=std::string::npos){  timeHours += 12;  }  this->sunsetSeconds = timeHours \* 60 \* 60 + timeMinutes \* 60 + timeSeconds;  return true;  } else  {  qDebug() << "parse time wrong status";  return false;  }  } else {  qDebug() << "parse time not object";  return false;  }  }  void SunData::setup(){  //http calls(one for sunrise sunset, other for utc shift)  //analyzing  //setting shift, sunrise, sunset, previous 2 in seconds, shift  // network getters  string timeJsonStr = askTime();  string utcJsonStr = askUtc();  if(timeJsonStr.compare("ERROR") == 0 || utcJsonStr.compare("ERROR") == 0 ){  // leaves all fields empty, after that class should be checked and not added to list  this->city = "ERROR";  return;  }  // json parsers : setts appropriate fields or leave them empty  if(!parseTimeJsonStr(timeJsonStr) ){  qDebug() << "ERRORS WITH PARSING TIME OCCURED";  this->city = "ERROR";  return;  }  if(!parseUtcJsonStr(utcJsonStr)){  qDebug() << "ERRORS WITH PARSING UTC OCCURED";  this->city = "ERROR";  return;  }  }  string SunData::askTime(){  string result;  try {  curlpp::Cleanup myCleanup;  string url = "https://api.sunrise-sunset.org/json?lat=";  url += std::to\_string(this->lat) ;  url += "&lng=" ;  url += std::to\_string(this->lng);  //qDebug() << QString::fromStdString(url);  curlpp::options::Url myUrl(url);  curlpp::Easy myRequest;  myRequest.*setOpt*(myUrl);  std::ostringstream os;  os << myRequest;  result = os.str();  } catch( curlpp::RuntimeError &e ) {  qDebug() << e.*what*();  return "ERROR";  } catch( curlpp::LogicError &e ) {  qDebug() << e.*what*();  return "ERROR";  }  // qDebug() << QString::fromStdString(result);  return result;  }  string SunData::askUtc(){  string result;  try {  curlpp::Cleanup myCleanup;  string url = "https://maps.googleapis.com/maps/api/timezone/json?location=";  url += std::to\_string((int)this->lat) ;  url += "," ;  url += std::to\_string((int)this->lng);  url += "&timestamp=";  url += std::to\_string(time(0));  url += "&key=AIzaSyA61GBAbkHvAjBkazo7Kuy7XYi\_1As7xOw";  qDebug() <<"url" << QString::fromStdString(url);  curlpp::options::Url myUrl(url);  curlpp::Easy myRequest;  myRequest.*setOpt*(myUrl);  std::ostringstream os;  os << myRequest;  result = os.str();  } catch( curlpp::RuntimeError &e ) {  qDebug() << e.*what*();  return "ERROR";  } catch( curlpp::LogicError &e ) {  qDebug() << e.*what*();  return "ERROR";  }  qDebug() << QString::fromStdString(result);  return result;  }  int SunData::GetSunriseSeconds(){  return this->sunriseSeconds;  }  int SunData::GetSunsetSeconds(){  return this->sunsetSeconds;  }  time\_t SunData::GetTimeSeconds(){  //getting local time - 2\*60\*60  //adding seconds according to UTCShift  //returning  time\_t timeUTCZero = time(0) - 3\*60\*60;  return timeUTCZero + this->UTCShiftSeconds ;  }  std::string SunData::GetLocalTime(){  //get time seconds method  //converting it to str  time\_t localtime = GetTimeSeconds();  return ctime(&localtime);  }  std::string SunData::GetSunrise(){  return sunrise;  }  std::string SunData::SunData::GetSunset(){  return sunset;  }  std::string SunData::GetTimeZoneId(){  return this->timeZoneId;  }  std::string SunData::GetTimeZoneName(){  return this->timeZoneName;  }  int SunData::GetUTCShiftSeconds(){  return this->UTCShiftSeconds;  }  time\_t SunData::GetTimeSinceMidnightSeconds(){  time\_t curTime = this->GetTimeSeconds();  struct tm tms;  time\_t t2;  localtime\_r(&curTime, &tms);  tms.tm\_hour = 0;  tms.tm\_min = 0;  tms.tm\_sec = 0;  t2 = mktime(&tms);  return curTime - t2;  }  std::string SunData::GetCity(){  return city;  }  float SunData::GetLat(){  return lat;  }  float SunData::GetLng(){  return lng;  }  std::string SunData::GetUTCShift(){  return std::to\_string(UTCShiftSeconds) ;  } |

|  |
| --- |
| sundata.h |
| #ifndef SUNDATA\_H  #define SUNDATA\_H  #include <string>  #include <qmetaobject.h>  using namespace std;  class SunData  {  private:  int analyzeStr();  void setup();  string askTime();  string askUtc();  bool parseUtcJsonStr(std::string str);  bool parseTimeJsonStr(std::string str);  std::string sunrise;  std::string sunset;  std::string timeZoneId;  std::string timeZoneName;  int sunriseSeconds;  int sunsetSeconds;  std::string city;  float lat;  float lng;  int UTCShiftSeconds;  public:  SunData();  SunData(string city, float latitude, float length);  time\_t GetTimeSeconds();  time\_t GetTimeSinceMidnightSeconds();  std::string GetLocalTime();  std::string GetSunrise();  std::string GetSunset();  std::string GetTimeZoneId();  std::string GetTimeZoneName();  int GetSunriseSeconds();  int GetSunsetSeconds();  std::string GetCity();  float GetLat();  float GetLng();  std::string GetUTCShift();  int GetUTCShiftSeconds();  };  Q\_DECLARE\_METATYPE(SunData )  #endif // SUNDATA\_H |

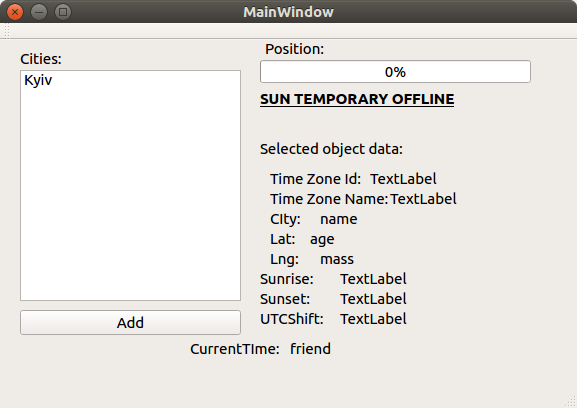
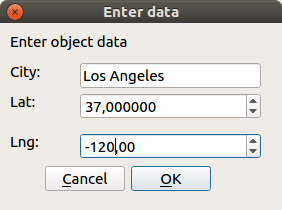
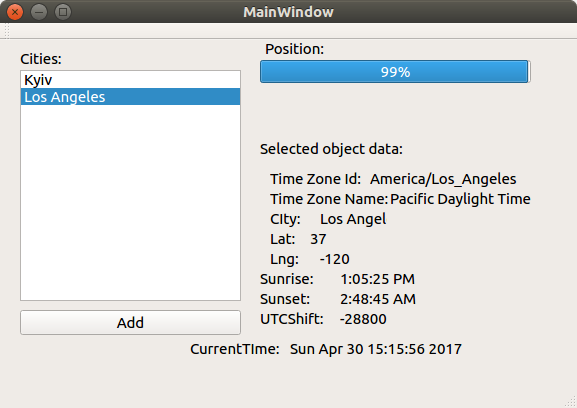
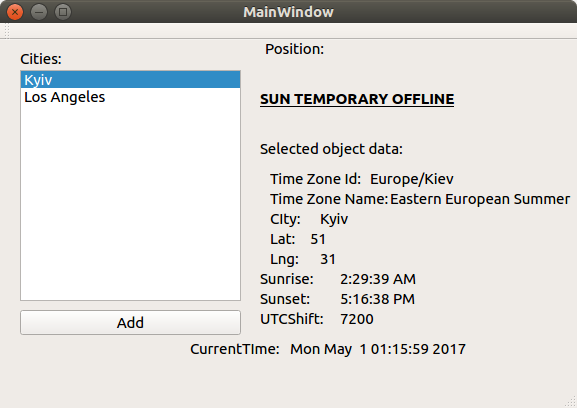
|  |
| --- |
| mainwindow.cpp |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "sundata.h"  #include <QString>  #include <QDebug>  #include "adddialog.h"  #include <QPainter>  #include <time.h>  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  SunData Kyiv("Kyiv", 51, 31);  if(Kyiv.GetCity().compare("ERROR") == 0){  qDebug() << "FAILED TO GET CITY DATA";  return;  }  QVariant qVariant;  qVariant.setValue(Kyiv);  QListWidgetItem \*qCityListItem = new QListWidgetItem();  qCityListItem->setText(QString::fromStdString(Kyiv.GetCity()));  qCityListItem->*setData*(Qt::*UserRole*, qVariant);  ui->City\_listWidget->addItem(qCityListItem);  }  MainWindow::~*MainWindow*()  {  delete ui;  }  void MainWindow::on\_City\_listWidget\_itemSelectionChanged()  {  auto items = ui->City\_listWidget->selectedItems();  if (items.size() == 0) {  return;  }  QListWidgetItem \* selectedItem = items[0];  SunData city = selectedItem->*data*(Qt::*UserRole*).value<SunData>();  ui->City\_label->setText(QString::fromStdString(city.GetCity()));  ui->CurTime\_label->setText(QString::fromStdString(city.GetLocalTime()));  ui->Lat\_label->setText(QString::number(city.GetLat()));  ui->Lng\_label->setText(QString::number(city.GetLng()));  ui->Sunrise\_label->setText(QString::fromStdString(city.GetSunrise()));  ui->Sunset\_label->setText(QString::fromStdString(city.GetSunset()));  ui->UTCShift\_label->setText(QString::fromStdString(city.GetUTCShift()));  ui->ZoneId\_label->setText(QString::fromStdString(city.GetTimeZoneId()));  ui->ZoneName\_label->setText(QString::fromStdString(city.GetTimeZoneName()));  //add something for showing circle as sun and turnong off/on sunavalaible lable  int dayLengthSeconds = abs((city.GetSunsetSeconds() + city.GetUTCShiftSeconds())%(24\*60\*60) -  (city.GetSunriseSeconds() + city.GetUTCShiftSeconds()) % (24\*60\*60));  time\_t thisDaySeconds = city.GetTimeSinceMidnightSeconds() -  abs((city.GetSunriseSeconds() + city.GetUTCShiftSeconds())%(24\*60\*60));  qDebug() << "dayLengthSeconds" << dayLengthSeconds << " thisDaySeconds" << thisDaySeconds <<  "time since midnight" << city.GetTimeSinceMidnightSeconds();  if(thisDaySeconds < 0 || thisDaySeconds > dayLengthSeconds){  ui->SunAvalaible\_label->*setVisible*(true);  ui->Sun\_progressBar->*setVisible*(false);  } else {  ui->SunAvalaible\_label->*setVisible*(false);  ui->Sun\_progressBar->*setVisible*(true);  ui->Sun\_progressBar->setValue(thisDaySeconds \* 100 / dayLengthSeconds);  }  }  void MainWindow::on\_Add\_pushButton\_pressed()  {  adddialog dialog;  dialog.setWindowTitle("Enter data");  if (dialog.*exec*()) {  QString city;  float lng;  float lat;  dialog.GetOptions(city, lat, lng);  SunData City(city.toStdString(), lat, lng);  if(City.GetCity().compare("ERROR") == 0){  qDebug() << "FAILED TO GET CITY DATA";  return;  }  QVariant qVariant;  qVariant.setValue(City);  QListWidgetItem \*qCityListItem = new QListWidgetItem();  qCityListItem->setText(QString::fromStdString(City.GetCity()));  qCityListItem->*setData*(Qt::*UserRole*, qVariant);  ui->City\_listWidget->addItem(qCityListItem);  qDebug() << "Accepted!";  } else {  qDebug() << "Rejected!";  }  } |

|  |
| --- |
| adddialog.cpp |
| #include "adddialog.h"  #include "ui\_adddialog.h"  adddialog::adddialog(QWidget \*parent) :  QDialog(parent),  ui(new Ui::adddialog)  {  ui->setupUi(this);  }  adddialog::~*adddialog*()  {  delete ui;  }  void adddialog::GetOptions (QString & city, float & lat, float & lng){  city = ui->City\_lineEdit->text();  lat = ui->Lat\_doubleSpinBox->value();  lng = ui->Lng\_doubleSpinBox->value();  } |

|  |
| --- |
| **adddialog.h** |
| **#ifndef ADDDIALOG\_H**  **#define ADDDIALOG\_H**  **#include <QDialog>**  **namespace Ui {**  **class adddialog;**  **}**  **class adddialog : public QDialog**  **{**  **Q\_OBJECT**  **public:**  **explicit adddialog(QWidget \*parent = 0);**  **void GetOptions (QString & city, float & lat, float & lng);**  **~*adddialog*();**  **private:**  **Ui::adddialog \*ui;**  **};**  **#endif // ADDDIALOG\_H** |

|  |
| --- |
| **main.cpp** |
| **#include "mainwindow.h"**  **#include <QApplication>**  **int main(int argc, char \*argv[])**  **{**  **QApplication a(argc, argv);**  **MainWindow w;**  **w.show();**  **return a.exec();**  **}** |

**Приклади результатів**



**Висновки**

Виконавши дану лабораторну роботу було створено tcp client та використано публічне api.