Міністерство освіти і науки України
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій

Кафедра комп’ютерних наук

Комплексне практичне індивідуальне завдання

з дисципліни «Конструювання програмного забезпечення»:

«REST - клієнт»

Виконав:
студент групи ПЗАС-31
Валігура Р. Д.
Перевірив:
Піговський Ю. Р

Тернопіль – 2014

Вступ

Серед нових проектів зараз все більшої популярності набувають веб API-сервіси. Як правило, API визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, які зазвичай у розширенні мови розмітки XML або в форматі об'єктного запису JavaScript (JSON). У той час як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-ресурсів та ресурсів-орієнтованої архітектури (ROA).

Таким чином Web-API є зручним та надійним рішенням для підтримки одного продукту на різних платформах (Web-сайти, мобільні додатки або інші типи додатків за необхідності).

При налагодженні свого API-сервісу я зіткнувся з проблемою, нестачі інструментарію. Бракувало утиліти яка б дозволяла надсилати запити до REST-API, а також вручну встановлювати заголовки запиту, його тіло, задавати авторизацію, тощо. Серед популярних додатків такого типу я знайшов розширення для Google Chrome – Postman, та схожі розширення для інших браузерів. Але Postman був занадто прожерливим для мого комп’ютера, і коли від сервера верталася дуже велика відповідь, просто вводив комп’ютер в стан недієздатності. Розширення в інших браузерах поступалися функціональністю. Тому, було прийнято рішення написати свій REST-клієнт.

Опис програми

Додаток призначений для відправки HTTP-запитів до сервера. Написаний він на мові програмування JavaScript використовуючи Qt Framework. Для побудови графічного інтерфейсу була використана декларативна мова розмітки QML.

На даний момент підтримуються наступні методи: GET, POST, PUT, DELETE. Планується підтримка методів OPTIONS, PATCH, HEAD та інших.

Підтримується ручне додавання заголовків запиту та можливість, введення тіла запиту, та заповнення даних для відправки форм.

Відповідь отриману від сервера можна подивитися в “сирому вигляді”, або обробленому для візуального сприйняття (форматовані JSON та XML). На випадок, коли вертається HTML, можна увімкнути режим браузерного перегляду. Даного функціоналу вдалося досягти за допомогою модуля WebKit, що включений до Qt Framework.

Лістинги коду

HTTPQuicker.pro

```
TEMPLATE = app
CONFIG += c++11
QT += qml quick

SOURCES += main.cpp
RESOURCES += qml.qrc

# Additional import path used to resolve QML modules in Qt Creator's code model
QML_IMPORT_PATH =

# Default rules for deployment.
include(deployment.pri)

HEADERS +=

DISTFILES += \
    lib/request.js
```

Workspace.ui.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtWebKit 3.0

Rectangle {
    property alias send: send
    property alias url: url.text
    property alias method: method.currentText
    property alias dataSend: dataSend.text
    property alias responseBody: response.text
   // property alias webView: webviewi

    width: 700
    height: 600

    Rectangle {
        id: tabBar
        width: parent.width
        height: 30
        border.color: "#d2d2d2"
        border.width: 1
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#ffffff" }
            GradientStop { position: 3.0; color: "#5a5a5a" }
        }
    }

    Rectangle {
        id: workspace
        x: 0
        y: tabBar.height
        width: parent.width
        height: parent.height - this.y
        Rectangle {
            id: rectangle1
            x: 0
            y: 0
            width: 208
            height: parent.height
            color: "#dfdfdf"
            radius: 0
            border.width: 1
            border.color: "#7b7b7b"

            Rectangle {
                id: rectangle4
                x: 0
                y: 0
                width: parent.width
                height: 31
                color: "#696969"
            }
        }
```

```
Rectangle {
    id: requestOptions
    x: 214
    y: 8
    width: parent.width - this.x - 8
    height: 127
    color: "#aee1a7"

    TextField {
        id: url
        x: 8
        y: 8
        width: 171
        height: 20
        placeholderText: qsTr("http://example.com")
        text: qsTr("")
    }

    ComboBox {
        id: method
        x: 8
        y: 34
        width: 76
        height: 23
        model: [ "GET", "POST", "PUT", "DELETE" ]
        activeFocusOnPress: false
    }

    Button {
        id: send
        x: 8
        y: 63
        text: qsTr("Send")
    }
}

ScrollView {
    x: 100
    y: 200
    width: 500
    height: 500
    WebView {
        id: webview
        z: 2
    }
}


TextArea {
    id: response
    x: 214
    y: 300
    width: parent.width - this.x - 8
    height: parent.height - this.y - 8
}

TextArea {
    id: dataSend
    x: 214
```

```
            y: 144

            width: parent.width - this.x - 8
            height: parent.height - response.height - this.y - 20
        }
    }
}
```

Main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QtQml>


int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

    return app.exec();
}
```

Main.qml

```
import QtQuick 2.4
import QtQuick.Window 2.2
import 'lib/request.js' as HTTP

Window {
    width: 700
    height: 600
    visible: true

    Workspace {
        id: ui
        anchors.fill: parent
        send.onClicked: {
            var Request = HTTP.request;
            Request.method = method;
            Request.addHeader("Content-Type", "application/json");
            Request.setBody(dataSend);
            Request.url = url;
            Request.send(function(response, success) {
                console.log(success);
                  ui.webView.loadHtml(response);
            });
        }
    }
}
```

Qml.qrs

```
<RCC>
    <qresource prefix="/">
        <file>main.qml</file>
        <file>Workspace.ui.qml</file>
        <file>lib/request.js</file>
    </qresource>
</RCC>
```

lib/request.js

```
var request = {
    method: 'GET',
    url: null,
    headers : [],
    body: null,

    addHeader: function(name, value) {
      this.headers.push({
         name: name,
         value: value
      });
    },

    setBody: function(body) {
        this.body = body;
      this.addHeader("Content-length", body.length);
    },

    send: function(action){
        var http = new XMLHttpRequest()

        if(this.method === 'GET' && this.body != '') {
            var query = JSON.parse(this.body);

            var str = Object.keys(query).map(function(key){
              return encodeURIComponent(key) + '=' +
encodeURIComponent(query[key]);
            }).join('&');

            this.url += ((this.url.search(/\?/g) == -1) ? "?": "&") +str;
        }

        http.open(this.method, this.url, true);

        this.headers.forEach(function(header, index, array){
            http.setRequestHeader(header.name, header.value);
        });

        http.onreadystatechange = function() {
            if(http.readyState == 4){
                if(http.status == 200) {
                    action(http.responseText.toString(), true);
                } else {
                    action("Cannot connect", false);
                }
            }
        }
        http.send(this.body);
    }
}
```

Висновок

Виконуючи комплексне практичне завдання, я закріпив навички проектування програмного забезпечення. Розробив додаток для взаємодію з REST-API сервісами. Закріпив навички роботи з системою контролю версій GIT.