MINISTERUL EDUCATIEI REPUBLICII MOLDOVA

UNIVERSITATEA TEHNICA A MOLDOVEI

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Inginerie Software si Automatica

RAPORT

Programarea în rețea

Lucrare de laborator Nr. 3

Tema: Protocolul HTTP.

Proiectare și programare aplicație client

A elaborat: st. gr. TI-142 Chifa Vladislav

A verificat: lector asistent Ostapenco Stepan

Chișinău 2017

Scopul lucrării

Protocolul HTTP. Clienţi şi servere Web. Metodele HTTP. Câmpuri HTTP. Componente .Net/Java pentru elaborare client HTTP. Rezultatul învăţării: abilităţi de utilizare a clienţilor Web, cunoştinţe în instalarea şi configurarea serverului Apache, abilităţi de creare aplicaţii client pentru servere HTTP.

Sarcina lucrării

a) Client simplu *http* apt să transmită și să primească răspunsuri la cereri tip Get,Head și POST;
b) Aplicație crawler de parcurgere conform unui algoritm a unui șir de resurse Webș;
c) Aplicații e colectare a unor informții specific pe diverse resurse Web;
d) Aplicație care validează hiperlegăturile prezente pe pagina resursei specificate;

Protocolul HTTP

Este un protocol la nivel aplicatie destinat sistemelor de informare distribuite, "colaborative", de genul hypermedia. Aparut ca protocol de baza pentru WWW înca din 1990, a cunoscut o serie de transformari, o versiune "finala" neexistînd nici în prezent. Versiunea cea mai folosita este înca 1.0, iar 515d36f versiunea 1.1 - compatibila "în jos" cu 1.0, dar aducînd îmbunatatiri în special în directia folosirii mai eficiente a resurselor - este pe cale sa se impuna ca nou standard. De aceea, o parte din aspectele care urmeaza nu trebuie privite ca referinte "batute în cuie", ci ca instantanee ale unei specificatii pe cale sa se nasca, extrase dintr-o schita, un "draft" care poate se va mai schimba mult.

Numele este acronimul pentru **HyperText Transfer Protocol,**desi la origine "hypertext" a fost definitoriu, practica curenta l-a dus destul de repede înspre "hypermedia" - documentele vehiculate cuprinzînd nu numai text, ci si sunet, imagine sau informatii structurate.

Aplicatiile care folosesc protocolul - cei doi parteneri în discutie, cele doua capete ale unei conexiuni - sînt entitati abstracte din punct de vedere al protocolului. Ele trebuie "doar" sa poata comunica între ele ceea ce înseamna, în principiu, posibilitatea de a primi sau formula cereri si de a formula sau receptiona raspunsuri, ca în celebrul exemplu al filozofilor ce vorbesc doua limbi diferite, folosind pentru comunicare translatorii care trimit mesajele filozofilor (traduse) prin intermediul postasilor. Nici un nivel nu se preocupa de celalalt.

Metodele protocolului HTTP

GET este una dintre cele mai importante metode si singura care era disponibila în prima versiune a protocolului, HTTP/0.9. GET este metoda care "aduce" ceva de la resursa; mai concret, daca resursa este un proces care produce date (o cautare de pilda), raspunsul la metoda GET va fi o entitate care sa cuprinda acele date. Raspunsul este unul singur: aceasta este o caracteristica de baza a protocolului. Chiar daca volumul de date care trebuie incluse în raspuns este mare, nu se face o fractionare în bucatele mai mici, care sa permita transferul mai usor al raspunsului. Din punct de vedere al protocolului HTTP, discutia este totdeauna simpla: o întrebare are un raspuns. Nu se pot pune mai multe întrebari pentru a obtine un singur raspuns, nu se pot formula mai multe raspunsuri la o întrebare.

HEAD este o metoda similara cu GET, folosita în principiu pentru testarea validitatii si/sau accesibilitatii unei resurse, sau pentru a afla daca s-a schimbat ceva. Sintaxa este similara metodei GET; spre deosebire de GET însa, datele eventual produse de resursa în urma cererii nu sînt transmise; doar caracteristicile acestora, si un cod de succes sau eroare. Ceva de genul "daca ti-as cere sa executi cererea mea, ce mi-ai raspunde?".

POST este metoda prin care resursei specificate în cerere i se cere sa îsi subordoneze datele incluse în entitatea care trebuie sa însoteasca cererea. Cu POST se poate adauga un fisier unui anumit director, se poate trimite un mesaj prin posta electronica, se poate adauga un mesaj unui grup de stiri, se pot adauga date unei baze de date existente, etc. Metoda POST este generala; care sînt procesele pe care un anumit server le accepta sau cunoaste îi sînt strict specifice.

PUT este o metoda care cere serverului ceva mai mult decît POST: sa stocheze/memoreze entitatea cuprinsa în cerere cu numele specificat în URI. Daca resursa specificata exista deja, entitatea noua trebuie privita ca o versiune modificata care ar trebui sa o înlocuiasca pe cea existenta. Serverul, bineînteles, va accepta sau nu aceasta cerere, functie de drepturile de acces pe care i le-a acordat clientului, si va raspunde cererii cu informatii corespunzatoare ("s-a facut", "nu pot", "nu ai voie sa faci treaba asta" etc.). Pentru a evita situatii care sa duca la încarcarea excesiva si nejustificata a retelei - de exemplu, un client care vrea sa "posteze" un text de 10 MB, fara sa tina seama de faptul ca serverul nu mai are atît loc atît o cerere de tipul POST cît si una de tipul PUT se desfasoara în doi timpi: întîi, clientul trimite numai parametrii metodei, fara sa trimita datele efective pe care le vrea postate. Dupa care asteapta 5 secunde. În acest timp, daca serverul raspunde, clientul ia în seama si analizeaza raspunsul serverului (iar daca acesta este "nu mai am loc", datele nu se mai transmit). Daca nu soseste nici un raspuns în timpul de asteptare, se considera implicit ca serverul accepta datele si acestea sînt transmise de catre client.

PATCH este o metoda similara lui PUT, dar nu contine toate datele care sa defineasca resursa, ci numai diferentele fata de versiunea existenta pe server. Cu toate informatiile necesare care sa îi permita serverului sa reconstruiasca o versiune la zi a resursei.

COPY, MOVE si **DELETE** sînt metode prin care se cere ca resursa specificata în URI-ul din cerere sa fie copiata în locatiile specificate ca parametri pentru metoda, mutata acolo sau respectiv doar stearsa.**LINK** si **UNLINK** sînt metode prin care resursa specificata în cerere este legata/dezlegata de alte resurse, stabilind una sau mai multe relatii cu acestea din urma, specificate ca parametrii pentru metoda. Ar putea fi de exemplu un index pentru o baza de date, un cuprins pentru un set de documente, etc.

TRACE este o metoda care îi permite clientului sa vada cum ajung cererile sale la server, pentru a verifica/diagnostica conexiunea, pentru a se verifica pe sine sau pentru a determina felul în care eventualele proxy-uri de pe parcurs au modificat cererea initiala. Serverul, în raspuns la aceasta cerere, va trimite în ecou cererile care îi vin de la client, fara sa le mai trateze ca cereri "reale".

WRAPPED este o metoda care "contrazice" principiul protocolului de a trimite totdeauna o singura cerere si a astepta un singur raspuns. Via WRAPPED, mai multe cereri, care în mod obisnuit ar fi succesive, sînt "împachetate" într-una singura. Iar o alta aplicare a metodei tinteste masuri de securizare - o cerere poate fi cifrata si transmisa prin metoda WRAPPED, ceea ce va determina serverul sa actioneze în doi pasi: întîi sa descifreze cererea reala, iar apoi sa îi dea curs acesteia.

În continuare am realizat o aplicație in care executăm aceste metode ale protocolului HTTP.În Figura 1 am realizat un Search Link , iar in figurile 1,2,3 sunt reprezentate metodele GET,POST și HEAD.

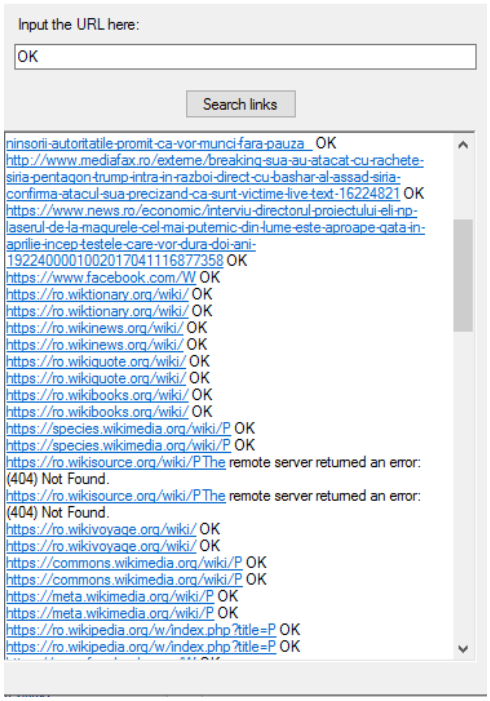

Figura 1 - Link cheker .

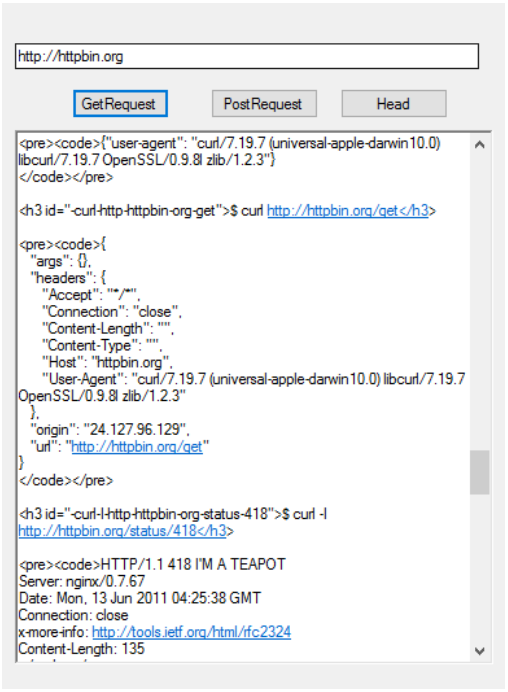

Figura 2 – Metode Get .

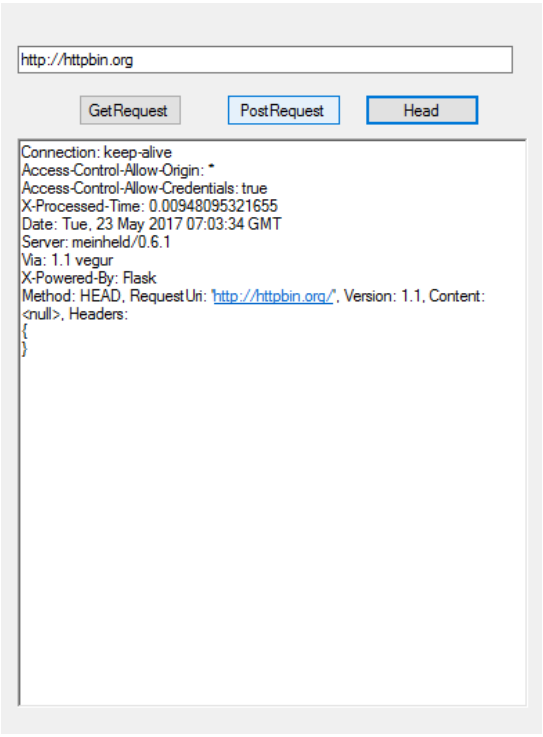

Figura 3 – Metoda HEAD.

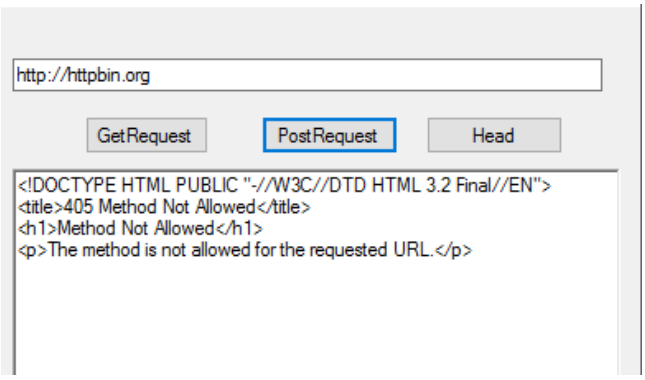

Figura 4 – Medota POST.

Concluzie

În aceată lucrare de labortor am lucrat biblioteca din Android Studio care ne oferă funcționalitatea cu ajutorul căreia putem crea cereri și răspunsuri conform protocolului HTTP.Protocolul HTTP este un protocol de tip cerere-răspuns cu ajutorul căruia putem trimite și primi date de la un server.Protocolul HTTP are 9 metode care asigură funcționalul necesar pentru a interacționa cu un server la nivel de aplicație.În această lucrare de laborator am utilizat numai metoda GET așa cum am avut nevoie numai de a primi date de pe server și nu de alte funcționalități .

Anexa:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text.RegularExpressions;
using System.Windows.Forms;

namespace PR_Lab3_Sample
{
     public partial class Form1 : Form
     {
          public Form1()
          {
               InitializeComponent();
          }


          private void searchBtn_Click(object sender, EventArgs e)
          {
               try
               {
                    searchBtn.Enabled = false;
                    searchRichTextBox.Text = "";

                    var req = WebRequest.Create(textBox1.Text);
                    var resp = req.GetResponse();

                    var dataStream = resp.GetResponseStream();

                    var reader = new StreamReader(dataStream);

                    var respFromServer = reader.ReadToEnd();

                    textBox1.Text = ((HttpWebResponse)resp).StatusCode.ToString();
                    var pattern = @"(((http|https|ftp)+\:\/\/)[&#95;.a-z0-9-]+\.[a-z0-
9\/&#95;:@=.+?,##%&~-]*[^.|\'|""|\# |!|\(|?,| |>|<|;|\)])";
                    var matches = Regex.Matches(respFromServer, pattern,
RegexOptions.Singleline);

                    foreach (var link in from Match match in matches select
match.Groups[1].Value)
                    {
                         try
                         {
                              var reqt = WebRequest.Create(link);
                              var respt = reqt.GetResponse();
                              searchRichTextBox.AppendText("\n" + link + " " +
((HttpWebResponse)respt).StatusCode);
                              respt.Close();
                         }
                         catch (Exception ex1)
                         {
                              searchRichTextBox.AppendText("\n" + link + ex1.Message);
                         }
                    }

                    searchBtn.Enabled = true;
               }
               catch (Exception ex)
```

```
            {
                MessageBox.Show(ex.Message);
                searchBtn.Enabled = true;
            }
        }

        private async void PostRequest(string url)
        {
            IEnumerable<KeyValuePair<string, string>> queries = new
List<KeyValuePair<string, string>>()
            {
                new KeyValuePair<string, string>("FirstPost", "hello"),
                new KeyValuePair<string, string>("SecondPost", "world")
            };
            HttpContent posContent = new FormUrlEncodedContent(queries);
            using (var client = new HttpClient())
            {
                using (var response = await client.PostAsync(url, posContent))
                {
                    using (var content = response.Content)
                    {
                        var myContent = await content.ReadAsStringAsync();
                        methodsRichTextBox.AppendText(myContent.ToString());
                        Console.WriteLine(myContent);
                    }
                }
            }
        }

        private void post_btn_Click(object sender, EventArgs e)
        {
            methodsRichTextBox.Clear();
            PostRequest(textBox2.Text);
        }
        private async void getHead(string url)
        {
            HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Head,
url);


            using (var client = new HttpClient())
            {

                using (var response = await client.SendAsync(request))
                {
                    using (var content = response.Content)
                    {


methodsRichTextBox.AppendText(response.Headers.ToString());

methodsRichTextBox.AppendText(response.RequestMessage.ToString());

                    }


                }


            }
        }
```

```
        private async void GetRequest(string url)
        {
            using (var client = new HttpClient())
            {
                using (var response = await client.GetAsync(url))
                {
                    using (var content = response.Content)
                    {

                        var mycontent = await content.ReadAsStringAsync();
                        methodsRichTextBox.AppendText(mycontent.ToString());


methodsRichTextBox.AppendText(response.Headers.ToString());
                        methodsRichTextBox.AppendText("Content-Type:" +
response.Content.Headers.ContentType.ToString() + "\n");

                        methodsRichTextBox.AppendText("Content length:" +
response.Content.Headers.ContentLength.ToString() + "\n");

methodsRichTextBox.AppendText(response.Headers.Upgrade.ToString());

                        // methodsRichTextBox.AppendText(response.ReasonPhrase);




                    }
                }
            }
        }

        private void get_btn_Click(object sender, EventArgs e)
        {
            methodsRichTextBox.Clear();
            GetRequest(textBox2.Text);
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }
        private void Head_Click(object sender, EventArgs e)
        {
            methodsRichTextBox.Clear();
            getHead(textBox2.Text);
        }

    }
}
```