Ministerul Educaţiei al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Ingineria Software și Automatică

Raport

Lucrare de laborator nr. 3
la disciplina: *Programarea în rețea*

Tema: Protocolul HTTP. Proiectare și programare aplicație client.

A verificat: lector asistent Ostapenco Stepan
lector asistent Latu Eugenia

A efectuat: st. gr. TI-142 Chirica Alexandru

Chișinău 2017

Scopul lucrării:

Studierea protocolului HTTP. Clienţi şi servere Web. Metodele HTTP. Câmpuri HTTP. Componente Python pentru elaborarea clienților HTTP.

Obiectivul lucrării:

De elaborat o aplicație "parser" care analizează profilul "Instagram" a unui utlizator, parsează HTML structura pentru a depista și afișa informația despre utilizator. Datele "cookie" primite de la server să fie salvate local. Să fie descărcate local toate pozele utilizatorului folosind "Threading" și sincronizarea firelor de executie să fie elaborată pe bază de "Semafor". Descărcarea fisierelor să se îndeplinească în 3 etape (S = numărul total de poze / 3). Analizăm răspunsul serverului si verificăm la errori sau pagini inexistente.

1. **HTTP – HyperText Transfer Protocol**

Protocolul de transfer utilizat pe Web este **HTTP** (HyperText Transfer Protocol, rom.:Protocol de Transfer al Hipertextului). Acesta specifică ce mesaje pot trimite clienţii către servere şi ce răspunsuri primesc înapoi. Fiecare interacţiune constă dintr-o cerere ASCII, urmată de un răspuns MIME [1].

1.2 Metode

Cu toate că HTTP a fost proiectat pentru utilizarea în Web, el a fost creat intenţionat mai general decât era necesar în perspectiva aplicaţiilor orientate pe obiecte. Pentru aceasta sunt suportate operaţiile, denumite **metode**, care fac mai mult decât cele care doar cer o pagină Web. Fiecare cerere constă din una sau mai multe linii de text ASCII, în care primul cuvânt din prima linie este numele metodei cerute. Metodele incorporate sunt listate în Fig. 1.

Metoda	Descriere
GET	Cerere de citire a unei pagini Web
HEAD	Cerere de citire a antetului unei pagini de Web
PUT	Cerere de memorare a unei pagini de Web
POST	Adăugarea la o resursă anume (de exemplu o pagină de Web)
DELETE	Ștergerea unei pagini de Web
TRACE	Tipărirea cererii care a sosit
CONNECT	Rezervat pentru o utilizare în viitor
OPTIONS	Interogarea anumitor opțiuni

Fig. 1. - Metode de cerere standard pentru HTTP.

Metoda ***GET*** cere serverului să trimită pagina (prin care noi înţelegem obiect, în cel mai general caz, dar în practică de obicei doar un fişier). Pagina este codată corespunzător în MIME. Marea majoritate a cererilor către servere Web sunt metode GET. Forma uzuală a metodei GET este*: GET fişier HTTP-1.1* unde *fişier* denumeşte resursa (fişierul) ce va fi adusă, şi 1.1 este versiunea de protocol utilizat.

Metoda ***HEAD*** cere doar antetul mesajului, fără să ceară şi pagina propriu-zisă. Această metodă poate să fie utilizată pentru a afla când s-a făcut ultima modificare, pentru a obţine informaţii pentru indexare, sau numai pentru a verifica corectitudinea unui URL.

Metoda ***PUT*** este inversa metodei GET: în loc să citească o pagină, o scrie. Această metodă permite crearea unei colecţii de pagini de Web pe un server la distanţă. Corpul cererii conţine pagina. Pagina poate să fie codificată utilizând MIME, caz în care liniile care urmează după PUT pot include *Content-Type* şi antete de autentificare, pentru a demonstra că într-adevăr cel care face cererea are dreptul de a realiza operaţia cerută.

Similară metodei PUT este metoda ***POST***. Şi ea conţine un URL, dar în loc să înlocuiască date existente, noile date se vor adăuga într-un mod generalizat. De exemplu, se poate transmite un mesaj la un grup de ştiri sau adăuga un fişier la un sistem de informare în reţea. În practică, nici PUT şi nici POST nu sunt utilizate prea mult.

DELETE realizează ce era de aşteptat: ştergerea unei pagini. Ca şi la PUT, autentificarea şi drepturile de acces joacă aici un rol important. Nu există nici o garanţie asupra succesului operaţiei DELETE, deoarece chiar dacă serverul doreşte să execute ştergerea, fişierul poate să aibă atribute care să interzică serverului HTTP să îl modifice sau să îl şteargă.

Metoda ***TRACE*** este pentru verificarea corectitudinii. Ea cere serverului să trimită înapoi cererea. Această metodă este utilă când cererile nu sunt procesate corect şi clientul vrea să ştie ce fel de cerere a ajuns de fapt la server.

Metoda ***CONNECT*** nu este utilizată în prezent. Este rezervată pentru utilizări ulterioare.

Metoda ***OPTIONS*** asigură o modalitate pentru client de a interoga serverul despre proprietăţile acestuia sau despre cele ale unui anumit fişier.

Fiecare cerere obţine un răspuns ce constă din linia de stare şi posibile informaţii suplimentare (de exemplu, o parte sau toată pagina Web). Linia de stare conţine un cod de stare de trei cifre, indicând dacă cererea a fost satisfăcută şi dacă nu, cauza. Prima cifră este utilizată pentru împărţirea răspunsurilor în cinci mari grupuri, ca în Fig.2.

Codurile 1xx sunt utilizate în practică foarte rar. Codurile 2xx indică tratarea cu succes a cererii şi conţinutul (dacă există) este returnat. Codurile 3xx spun clientului să caute în altă parte, prin folosirea unui URL diferit, sau în propria memorie ascunsă.

Codurile 4xx indică insuccesul cererii din cauza unei erori la client, precum o cerere invalidă sau o pagină inexistentă. În fine, erorile 5xx indică o problemă în server, datorată codului său sau unei supraîncărcări temporare.

Cod	Semnificație	Exemple
1xx	Informație	100 = serverul acceptă tratarea cererii de la client
2xx	Succes	200 = cerere reușită; 204 = nu există conținut
3xx	Redirectare	301 = pagină mutată; 304 = pagina din memoria ascunsă este încă validă
4xx	Eroare la client	403 = pagină interzisă; 404 = pagina nu a fost găsită
5xx	Eroare la server	500 = eroare internă la server; 503 = încearcă mai târziu

Fig.2. - Grupuri de răspunsuri ale codurilor de stare.

1.3 Antete de mesaje

Linia de cerere (de exemplu linia cu metoda GET) poate fi urmată de linii adiţionale cu mai multe informaţii. Acestea poartă numele de antete de cerere. Această informaţie poate fi comparată cu parametrii unui apel de procedură. Răspunsurile pot avea de asemenea antete de răspuns. Anumite antete pot fi folosite în orice sens [2]. O selecţie a celor mai importante este dată în Fig.3.

Antetul ***User-Agent*** permite clientului să informeze serverul asupra programului său de navigare, sistemului de operare şi altor proprietăţi.

Antet	Tip	Descriere
User-Agent	Cerere	Informație asupra programului de navigare și a platformei
Accept	Cerere	Tipul de pagini pe care clientul le poate trata
Accept-Charset	Cerere	Seturile de caractere care sunt acceptabile la client
Accept-Encoding	Cerere	Codificările de pagini pe care clientul le poate trata
Accept-Language	Cerere	Limbajele naturale pe care clientul le poate trata
Host	Cerere	Numele DNS al serverului
Authorization	Cerere	O listă a drepturilor clientului
Cookie	Cerere	Trimite un cookie setat anterior înapoi la server
Date	Ambele	Data și ora la care mesajul a fost trimis
Upgrade	Ambele	Protocolul la care transmițătorul vrea să comute
Server	Răspuns	Informație despre server
Content-Encoding	Răspuns	Cum este codat conținutului (de exemplu, gzip)
Content-Language	Răspuns	Limbajul natural utilizat în pagină
Content-Length	Răspuns	Lungimea paginii în octeți
Content-Tzpe	Răspuns	Tipul MIME al paginii
Last-Modified	Răspuns	Ora și data la care pagina a fost ultima dată modificată
Location	Răspuns	O comandă pentru client pentru a trimite cererea în altă parte
Accept-Ranges	Răspuns	Serverul va accepta cereri în anumite limite de octeți
Set-Cookie	Răspuns	Serverul vrea să salveze un cookie la client

Fig. 3. - Câteva antete de mesaje HTTP.

Cele patru antete *Accept* spun serverului ce este dispus clientul să accepte în cazul în care acesta are un repertoriu limitat despre ceea ce este acceptabil. Primul antet specifică ce tipuri MIME sunt acceptate (de exemplu, text/html).

Al doilea reprezintă setul de caractere (de exemplu ISO-8859 sau Unicode-1-1). Al treilea se referă la metode de compresie (de exemplu, gzip). Al patrulea indică un limbaj natural (de exemplu, spaniola). Dacă serverul are mai multe pagini din care poate să aleagă, el poate utiliza această informaţie pentru a furniza clientului pagina pe care o caută. Dacă nu poate satisface cererea, este întors un cod de eroare şi cererea eşuează.

Antetul *Host* denumeşte serverul. El este luat din URL. Antetul este obligatoriu. Este utilizat deoarece anumite adrese IP pot servi mai multe nume de DNS şi serverul are nevoie de o anumită modalitate de a spune cărui calculator să-i trimită cererea.

Antetul *Authorization* este necesar pentru protecţia paginilor. În acest caz, clientul trebuie să demonstreze că are dreptul de a vedea pagina cerută. Acest header este utilizat în acest scop.

Antetul *Cookie* este utilizat de clienţi pentru a întoarce serverului un cookie care a fost anterior trimis de o maşină aflată în domeniul serverului.

Antetul *Date* poate fi utilizat în ambele sensuri şi conţine ora şi data la care a fost trimis mesajul.

Antetul *Upgrade* este folosit pentru a face mai uşoară crearea unei tranziţii către o viitoare (posibil incompatibilă) versiune a protocolului HTTP. Acesta permite clientului, să anunţe ce anume suportă, şi serverului să afirme ceea ce foloseşte.

Acum am ajuns la antetele utilizate exclusiv de către server în răspunsul cererilor. Primul, *Server*, permite serverului să spună cine este şi câteva proprietăţi, dacă doreşte.

Următoarele patru antete, toate începând cu *Content-*, permit serverului să descrie proprietăţile paginii pe care o transmite.

Antetul *Last-Modified* spune când a fost modificată ultima dată pagina. Acest antet joacă un rol important în mecanismul de memorie ascunsă.

Antetul *Location* este utilizat de server pentru a informa clientul că ar trebui să utilizeze un alt URL. Acesta poate fi folosit dacă pagina a fost mutată, sau pentru a da permisiunea mai multor URL-uri de a referi aceeaşi pagină (posibil pe servere diferite). Este de asemenea utilizată pentru companiile care au o pagină de Web principală în domeniul com, dar care redirecţionează clienţii la o pagină naţională sau regională în funcţie de adresa lor IP sau limba preferată.

Dacă o pagină este foarte mare, un client mic poate nu o doreşte dintr-o dată. Unele servere acceptă cereri în anumite intervale de octeţi, astfel că pagina poate fi citită în mai multe unităţi mai mici. Antetul *Accept-Ranges* anunţă asentimentul severului de a trata acest tip de cerere parţială de pagini.

Al doilea antet pentru cookie, *Set-Cookie*, se referă la modul în care serverele trimit cookie-uri la clienţi. Este de aşteptat salvarea cookie-ului de către client şi returnarea acestuia la cereri ulterioare ale serverului.

2. **Mersul lucrării**

Idea de bază a programul este de a realiza un parser care va primi de la utilizator un nickname de pe Instagram ca mai apoi programul să verifice existența utilizatorului introdus, să afișeze datele utilizatorului și să descarce local pozele utilizatorului.

Codul sursă: https://github.com/kivlara-mujik/PR1/blob/master/Lab-3/main.py

2.2 **Biblioteci externe folosite**

Bibliotecile necesare pentru realizarea acestui laborator au fost deja instalate în laboratorul numărul 1. Sărim pasul de instalarea a dependențelor.

- Grab – HTTP client. Biblioteca ne permite executarea diferitor cereri către server [3]. Setarea bibliotecii este dinamică, ne permite să modificăm fiecare antet din header-ul trimis către server.
- BeautifulSoup – folosim biblioteca dată la parsarea mai eficientă a structurii HTML [4]. Structura paginii returnată de către server poate să nu coincidă la standartele HTML sau să conțină unele greșeli de sintaxă. Biblioteca BeautifulSoup pregătește în mod automat structura pentru a fi mai ușor de manipulat și de a extrage datele necesare.

2.3 **Setarea HTTP clientului**

Pentru a îndeplini cererile către server, biblioteca Grab nu are nevoie de setări, ele deja sunt pre-setate în momentul instalării bibliotecii. Setările adăugate:

- user_agent – Numele HTTP Client-lui nostru. Acest nume este văzut de către server în moment ce clientul îndeplinește o oarecare cerere [5].
- cookiefile – Declarăm bibliotecii Grab adresa către fișierul unde vor fi stocate datele „cookie” primite din partea server-lui.

```
g = Grab(
    user_agent = 'PR-Lab3 CoolBrowser V1.0.0',
    cookiefile = 'cookie.txt'
)
```

Fig. 4. - Setarea HTTP Client-lui Grab.

2.4 **Analizăm răspunsul serverului**

După cum am menționat în Fig. 2. din partea serverului obținem codul stării paginii în urma execuției a cererii din partea HTTP Clientului.

```
nickname = raw_input('Instagram Nickname: ')

# Îndeplinim un GET Request la pagina de isntagram.
work = g.go('https://www.instagram.com/%s' % nickname)

# Analizăm răspunsul paginii.
if work.code == 404:
    print 'Eroare 404 - Utilizatorul nu există!'
else:

    # Verificăm erorile.
    if work.code != 200:
        print 'Greșeală la nivel de server...'
        exit()

    print '-' * 40
    print 'Utlizatorul "%s" există\n' % nickname
```

Fig. 5. - Analiza codului stării paginei.

În urma execuției programului, se solicită să fie indicat numele utilizatorului datele căruia vor fi parsate. Prin intermediul metodei „go” al bibliotecii „Grab” este accesată pagina de profil a utilizatorui indicat. HTTP Clientul returnează codul stării în parametrul „code”.

În caz că răspunsul paginii este greșeală de tipul „404” afișăm răspunsul că utilizatorul nu există și finisăm lucrul programului. Programul va fi forțat finisat dacă codul stării este diferit de „200 – OK”.

2.5 **Analiza structurii HTML**

Bilioteca „Grab” returnează răspunsul serverului cu structura paginii HTML. Acest răspuns îl trimitem ca argument bibliotecii BeautifulSoup care la rîndul ei structurază HTML-ul după un unic standard pentru a fi mai eficientă manipularea elementelor.

2.6 **Datele utilizatorului**

Datele despre profilul utilizatorului sunt stocate în corpul paginii într-o javascript variabilă globală sub numele „_sharedData”. Variabila este un array JSON cu toată informația utilizatorului și lista de fisiere (video, poze) în profil.

Cu ajutorul bibliotecii „re” (Regular Expresion) am parsat continutul variabilei „_sharedData” și am transormat text-ul JSON într-un Python array cu ajutorul bibliotecii „json”.

2.7 **Stocarea datelor „Cookie”**

În unul din pașii de setarea a HTTP Clientului am indicat adresa către fișierul „cookie.txt” în acest fisier biblioteca Grab va stoca datele cookie primite de la server. În *Fig. 6.* este reprezentat răspunsul serverului și datele „cookie” care le primim din partea serverului. În partea dreaptă este reprezentată structura a datelor cookie, în așa formă o putem întîlni și în browser-le moderne.

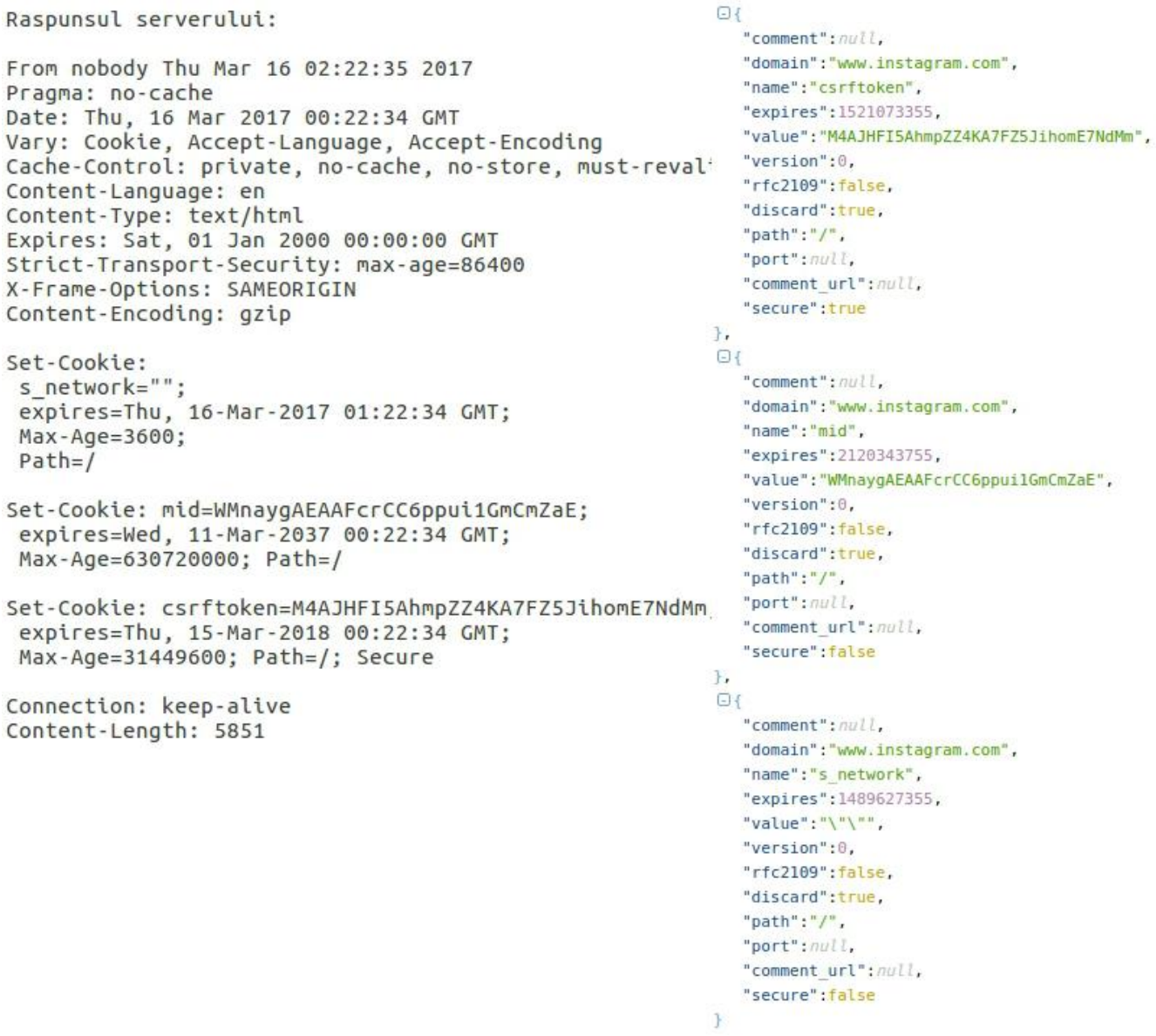

Fig. 6. - Răspunsul serverului și vizualizarea structurii datelor cookie.

Datele cookie sunt salvate în format JSON. Datele Cookie pot fi modificate manual cît și cu ajutorul bibliotecii Grab. Modificarea datelor cookie ne permite manipularea răspunsurilor serverului. În unul din cookie poate fi salvată limba curentă a paginei. Modificarea valorii a acesteia va duce la modificarea limbii la următorul request.

2.8 **Sincronizarea firelor de execuție**

Pentru a îmbunătăți viteza de execuție a programului am decis să îndeplinim descărcarea fisierelor în 3 etape. Spre exemplu, dacă numărul total de imagini este 9 atunci avem nevoie de un semafor care va permite executarea a 3 fire concomitent.

S = 9 / 3

Pentru îndeplinirea condiției am folosit semaforul “BoundedSemaphore” în care putem să indicăm valoarea inițială “S” care se va micșora cu o unitate de fiecare dată cînd un fir de execuție va intra în zona de control, în cazul dat zona de control este momentul de descărcare a imaginii. Valoarea “S” fiind 3 atunci doar 3 fire de excuție vor descărca 3 imagini paralel. După ce un fir de execuție a îndeplinit condiția el eliberează rîndul și un alt fir de execuție începe lucrul.

2.9 **Imaginile salvate**

Toate imaginile din profilul utilizatorului sunt salvate în mapa “images”. Fiecare imagine are numărul său unical în dependență de rîndul ei în listă. Imaginile sunt descărcate cu ajutorul bibliotecii standarte Python “**urllib**”.

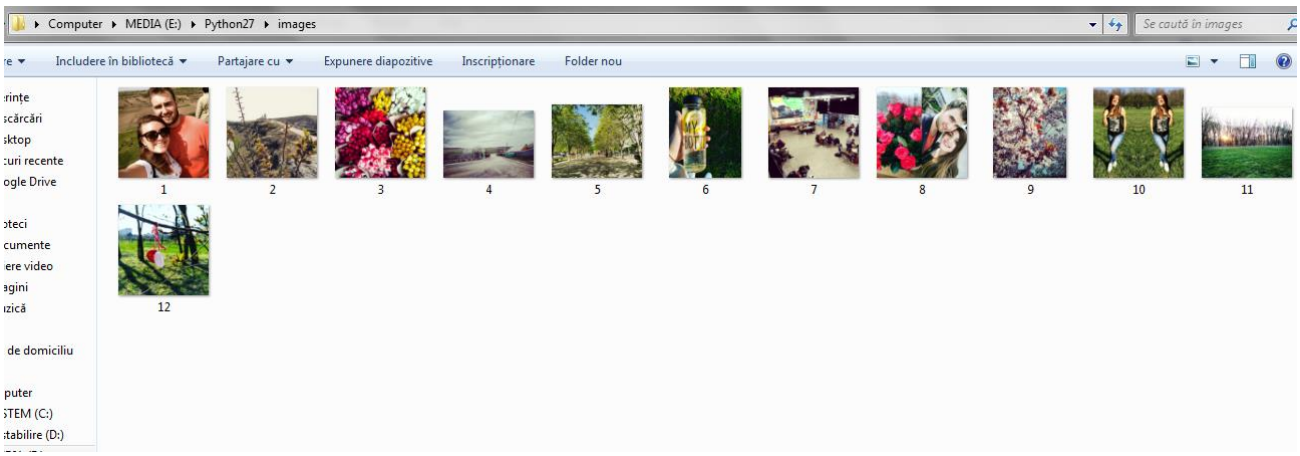

Fig. 7. - **Imaginile salvate**

3. **Rezultatul programului**

Pentru a observa lucrul semaforului a fost adăugată o “așteptare” artificială de 8 secunde pentru a observa cum are loc îndeplinirea a cîte 4 fire de execuție concomitent și celelalte fire sunt puse în așteptare.

```
======================== RESTART: E:\Python27\main.py ========================
Instagram Nickname: diiankkaa
---------------------------------------
Utlizatorul "diiankkaa" exista

ID: 1660840521
Numele: Diana Gurschi
Urmareste: 271 utilizatori
Îl urmaresc: 311 utilizatori
---------------------------------------
Raspunsul serverului:

From nobody Mon May 01 19:05:42 2017
Content-Type: text/html
X-Frame-Options: SAMEORIGIN
Cache-Control: private, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Vary: Cookie, Accept-Language, Accept-Encoding
Content-Language: en
Content-Encoding: gzip
Date: Mon, 01 May 2017 16:05:42 GMT
Strict-Transport-Security: max-age=86400
Set-Cookie: rur=ATN; Path=/
Set-Cookie: s_network=""; expires=Mon, 01-May-2017 17:05:42 GMT; Max-Age=3600;
 Path=/
Set-Cookie: csrftoken=ZjiU7c6Uqk1jiyGwAKAsSfia0GRwidd4;
 expires=Mon, 30-Apr-2018 16:05:42 GMT; Max-Age=31449600; Path=/; Secure
Connection: keep-alive
Content-Length: 4972


---------------------------------------
Numarul de fisiere: 242
Descarcam primele: 12 imagini
Info: Concomitent se vor descarca 4 imagini.


Descarcam imaginea numarul:Descarcam imaginea numarul:Descarcam imaginea numarul:Descarcam imaginea numarul:
>>> 1234
```

Fig. 8. - **Exemplu de lucru a programului.**

Concluzii

În era dezvoltării rapide a internetului capacitatea de analiză a traficului și modalitățile de manipularea cu protocoalele este un avantaj important pentru un inginer contemporan. În urma analizei răspunsurilor „header” din partea serverului am observat că proiectele mari nu se „împart” cu unele informații, spre exemplu „Denumirea serverului”, „Versiunea serverului” toate acestea sunt făcute din considerente a securitării.

Biblioteca HTTP Client Grab a ușurat elaborarea și înțelesul programului datorită simplității sale și unei documentații avansate. Unele probleme apărute în urma folosirii acestei biblioteci au fost rezolvate pe pagina oficială a bibliotecii pe github.

Proiectul „Instagram” are API, inițial programul era îndreptat spre folosirea acestui API și în cazul acesta nu mai era necesar să parsăm manual conținutul paginii ci doar de a apela o metodă specială care ar returna toate datele. Însă, politica de dezvoltarea a proiectelor ca „Facebook”, „Instagram” reduc din unele posibilități a funcționalului API sau folosirea lui necesită o autentificarea mai sofisticată cu dreptui confirmate cum ar fi oAuth2.

Manipularea datelor din „cookie” reprezintă un mare pericol pentru securitate dacă administratorul nu a analizat și nu a filtrat datele de intrare din partea utilziatorului. Toate datele „cookie” sunt stocate în calculator la client și clientul le poate modifica, spre exemplu ID-ul curent de logare, în așa caz (dacă securitatea nu prevede așa atac) utilizatorul poate prelua acesul administratorului.

Sincronizarea firelor de execuție și mai ales folosirea semaforului care ne permite modificarea numărului de fire executate paralel, a ajutat la înțelegerea de lucru a acestui mecanism. Viteza de execuție și de finisare a programui s-a îmbunătățit.

Bibliografie

1. Protocolul HTTP [Sursă electronică]

http://www.scritub.com/stiinta/informatica/html/Protocolul-HTTP71532.php 29 martie 2017

2. HTTP headers [Sursă electronică]

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers 10 martie 2017

3. Grab's documentation [Sursă electronică]

http://docs.grablib.org/en/latest/ 17 martie 2017

4. Beautiful Soup Documentation [Sursă electronică]

https://www.crummy.com/software/BeautifulSoup/bs4/doc/ 17 martie 2017

5. User Agent [Sursă electronică]

https://www.howtogeek.com/114937/htg-explains-whats-a-browser-user-agent/ 17 martie 2017