Laboratorul 10 - Protocolul HTTP

Responsabili: Radu-Ioan Ciobanu, Alexandru Hogea, Silviu Pantelimon

Obiective

In urma parcurgerii acestui laborator, studentul va fi capabil sa:

- inteleaga cum functioneaza protocolul HTTP
- inteleaga rolul entitatilor implicate in schimb de mesaje HTTP
- scrie un client simplu de HTTP
- interactioneze cu un API public

Contextul laboratorului

Daca acum 10-15 ani aplicatiile si serviciile web nu erau atat de raspandite, astazi sunt prezente pretutindeni. De exemplu, numarul website-urilor a ajuns sa fie aproximativ 2 miliarde si website-urile reprezinta o parte din web.

Pentru a putea patrunde in acest domeniu vast, este nevoie sa intelegem protocolul temelie al web-ului si anume, **HTTP**.

Protocolul HTTP

Informatii generale

HTTP (HyperText Transfer Protocol) este un protocol de nivel 7 (aplicatie) folosit pentru transferul informatiilor in internet. Este un protocol care opereaza peste date de tip ASCII.

La baza protocolului HTTP stau conceptele de **cerere** si **raspuns**. In cazul comunicatiei HTTP, o entitate inainteaza o cerere si cealalta trebuie, obligatoriu, sa ofere un raspuns.

HTTP functioneaza implicit peste portul 80. Versiunea securizata de HTTP, HTTPS, functioneaza implicit peste portul 443. Un server, insa, poate fi configurat sa asculte cereri HTTP pe orice port disponibil.

Cereri HTTP

Cu totii sunteti familiari cu acest format:

Exemplul de mai sus cuprinde:

- versiunea protocolului
- host-ul interlocutorului
- calea de pe serverul interlocutorului unde se va desfasura actiunea
- parametri aditionali de cerere (optionali)

Ce este prezentat in poza nu este o cerere HTTP, ci preambul unei cereri HTTP. De fapt, in momentul in care se da enter, browserul (sau orice alt client) creaza, bazat pe informatiile oferite, cererea HTTP efectiva.

Formatul cererii este urmatorul:

```
METODA CALE VERSIUNE_PROTOCOL\r\n
Host: HOST\r\n
Header1: Valoare Header1\r\n
Header2: Valoare Header2\r\n
...
Cookie: cheie1=valoare1; cheie2=valoare2; ...; cheieN=valoareN\r\n
\r\n
DATA
```

Linia de start contine 3 elemente.

Primul element este metoda folosita. Metodele HTTP sunt verbe ce descriu actiunea ce va fi efectuata asupra entitatii catre care se transmite cererea. Cele mai des utilizate cereri sunt:

- **GET** - interogare de resurse
- **POST** - aduagare de resurse. De obicei are si date atasate.
- **PUT** - modificare de resurse. De obicei are si date atasate.
- **DELETE** - stergere de resurse

Al doilea element este reprezentat de calea si parametrii de cerere (daca exista) unde se va actiona asupra resursei, pe server. In cazul in care exista parametri de cerere, acestia trebuie separati de restul cali prin **?**.

Al treilea element este reprezentat de versiunea protocolului de HTTP folosita. Implicit, din motive de securitate, este folosit HTTPS. Pentru varianta ne-securizata, ultima versiune este HTTP/1.1.

A doua linie descrie host-ul entitatii unde va fi transmisa cererea. Host-ul poate sa fie atat un *ip* cat si un *domeniu*.

Host-ul este, de fapt, tot un header. Pentru a omite aceasta linie este necesar sa puneti host-ul in cale, exact cum se scrie in browser. Totusi, este indicat sa tratati host-ul ca linie separata.

Headerele sunt scrise cate unul pe rand. Acestea sunt folosite pentru a transmite informatii aditionale catre interlocutor. Headerele se impart in 3 categorii:

- **Request Headers** - descriu modul in care se face cererea si cum se poate raspunde la ea. Exemplu: *User-Agent*
- **General Headers** - descriu informatii cu caracter general care tin de comunicare. Exemplu: *Keep-Alive*
- **Entity Headers** - descriu informatii despre datele (daca exista) atasate cererii. Exemplu: *Content-Type* si *Content-Length*

Daca exista data atasata cererii, trebuie specificate, obligatoriu, cele doua headere **Content-Type** si **Content-Length**

Cookies sunt scrise inlantuit, delimitat de punct si virgula (mai putin ultima). Implicit, comunicarea HTTP este considerata stateless. Nu se poate face corelatie intre oricare doua cereri succesive. Cookies retin bucati de informatie trimise de la server catre client, pentru a putea fi refolosite in cereri ulterioare.

Inaintea datelor (sau la finalul cererii, daca nu exista date) se pune intotdeauna \r\n

Data variaza in functie de tipul de data transmis. Cele mai des intalnite tipuri de date transmise sunt:

- **text/html** - De exemplu, pagini HTML
- **application/x-www-form-urlencoded** - Date de forma *key1=value1&key2=value2&...&keyN=valueN*. Datele sunt inlantuite prin "&"
- **application/json** - Date de forma JSON (Javascript Object Notation). Folosite des in interactiunea cu API-uri
- **multipart/form-data** - Date binare, de exemplu, fisiere

Pe baza informatiilor prezentate mai sus, o varianta **simplificata** a cererii catre facebook, din exemplu, ar arata asa:

```
GET /search/top/?q=programare%20web%202020 HTTPS\r\n
Host: facebook.com\r\n
User-Agent: Mozilla/5.0\r\n
Connection: keep-alive\r\n
Cookie: c_user=XXXXXXXXX; presence=XXXXXXX\r\n
\r\n
```

Este obligatoriu sa puneti \r\n la finalul fiecarui rand din cerere, cu exceptia datelor atasate

Raspunsuri HTTP

Orice cerere HTTP este urmata de un raspuns. Raspunsurile seamana cu cererile din punct de vedere al organizarii. Formatul este urmatorul:

```
PROTOCOL_VERSION STATUS_CODE STATUS_TEXT\r\n
Header1: Valoare Header1\r\n
Header2: Valoare Header2\r\n
...
HeaderN: Valoare HeaderN\r\n
Set-Cookie: cheie1=valoare1\r\n
Set-Cookie: cheie2=valoare2\r\n
...
Set-Cookie: cheieN=valoareN\r\n
\r\n
DATA
```

Linia de start contine 3 elemente.

Primul element este reprezentat de versiunea protocolului de HTTP folosit pentru a se raspunde.

Al doilea element este reprezentat de **statusul** raspunsului. Statusul este corelat de reusita, respectiv esecul cererii si de ce s-a intamplat pe entitatea catre care s-a trimis cererea. Exemplu de statusuri des intalnite:

- **200** - OK
- **201** - Created
- **204** - No Content
- **400** - Bad Request
- **401** - Unauthorized
- **403** - Forbidden
- **404** - Resource Not Found
- **500** - Internal Server Error

Al treilea element descrie textul care insoteste statusul.

Headerele urmeaza aceeasi structura si descriu acelasi lucru ca si in cazul cererilor.

Cookies sunt setate cate una pe linie. In afara de *cheie=valoare*, acestea mai au o serie de atribute atasate, precum **secure**, **httpOnly**, **domain**.

Data urmeaza aceeasi structura ca si in cazul cererilor.

Suportul de laborator

Va oferim acest suport pentru realizarea unui client HTTP scris in C. Aveti deja implementata partea de realizare de conexiune, partea de trimitere, respectiv receptionare bytes de la server in *helpers.c* si *buffer.c*.

Fisierele in care voi veti lucra sunt **client.c** si **requests.c**. In requests va trebui sa implementati scrierea cererilor de tip GET si POST si in client trebuie sa trimiteti cererea proaspat compusa catre destinatie.

Veti interactiona cu doua servere:

- Serverul scris de noi aflat la adresa *ec2-3-8-116-10.eu-west-2.compute.amazonaws.com* pe portul *8080*
- API-ul oferit de Openweather Map aflat la adresa *api.openweathermap.org* pe portul *80*

Exercitii

Exercitii Principale

Primele doua exercitii valideaza daca ati implementat corect functiile care compun mesajele pentru POST si GET:

1. GET pe adresa */api/v1/dummy* **(2p)**

2. POST cu orice data de forma **application/x-www-form-urlencoded** pe adresa */api/v1/dummy* **(2p)**

Urmatoarele exercitii valideaza daca ati inteles mecanismul de sesiune:

3. POST pe adresa */api/v1/auth/login* cu **username** student si **password** student. Data trebuie sa fie de forma **application/x-www-form-urlencoded (2p)**

4. GET pe adresa */api/v1/weather/key* (folosind cookie-ul obtinut la pasul precedent, care poate fi **hardcodat**) pentru a obtine o cheie de acces **(2p)**

5. GET pe adresa */api/v1/auth/logout* **(2p)**

Bonus

1. Faceti serverul sa returneze "Already logged in!" in momentul in care dati login **(0.5p)**

2. GET **catre API-ul de la openweather** pe adresa */data/2.5/weather* cu parametri de cerere **lat**, **lon** si **appid**. Appid trebuie sa contina cheia primita la exercitiul 4. Pentru simplitate, puteti *hardcoda* cheia (o preluati manual cu copy-paste) **(1p)**

Exercitiul 2 va fi punctat daca veti reusi sa primiti orice in afara de eroare

Verificare Bonus (nepunctat)

Pentru a valida informatia primita de pe API-ul celor de la OpenWeather v-am pus la dispozitie urmatoarea ruta:

POST pe adresa */api/v1/weather/:lat/:long* cu datele despre vreme primite de la exercitiul bonus 2. Pentru extragerea datelor puteti folosi functia *basic_extract_json_response* pe intreg raspunsul primit. In loc de **:lat** si **:long** trebuie sa puneti latitudinea si longitudinea folosita la exercitiul anterior. Data trebuie sa fie de forma **application/json**