3. HTTP server

Fokus ovih vježbi jest primjeniti naučeno na prethodnim vježbama u izradi jednostavnog HTTP servera. HTTP server mora moći primiti zahtjev i vratiti HTML stranicu kao rezultat.

3.1 HTTP server

Http server kojega radimo na vježbama treba moći poslati web stranicu klijentu koji ju je zatražio. Komunikacija između klijenta i servera ide HTTP protokolom poštujući specifikacije istog. Naravno da server neće podržavati sve funkcionalnost koje serveri danas imaju, ali će pomoći razumijevanju na koji način funkcioniraju.

3.2 HTTP zahtjev

HTTP zahtjev je definiran u dokumentu RFC2616 koji je dostupan svima preko internetu. HTTP protokol dozvoljava različite tipove zahtjeva, a mi ćemo se fokusirati (i ograničiti) na implementaciju GET-a. U sekciji 5 navedenog dokumenta se nalaze informacije koje su potrebne da implementiramo naš HTTP parser.
Parser je program koji nam služi za analizu teksta, bilo koja poruka koju naš server primi će biti u tom obliku. Stoga nam je potreban parser kako bismo analizirali sekcije HTTP zahtjeva kojeg je server primio. Naravno, neke detalje HTTP zahtjeva ćemo izostaviti, ali ćemo napraviti dovoljno da možemo razumjeti što je to i kako se njime barata.

Izgled HTTP zahtjeva:

Primjer 3.1: Izgled HTTP zahtjeva

```
1         Request        = Request-Line                ; Section 5.1
2                          *(( general-header           ; Section 4.5
3                           | request-header            ; Section 5.3
4                           | entity-header ) CRLF)     ; Section 7.1
5                          CRLF
6                          [ message-body ]             ; Section 4.3
```

Dakle, zahtjev se sastoji od: zahtjevne linije (engl. *Request-Line*), opcionalnog niza zaglavlja, *carriage return line feed* (CRLF - označava novu liniju) i opcionalnog tijela poruke. Svako zaglavlje opcionalnog niza zaglavlja završava s CRLF-om. Kako ovo implementirati?
Kako bi se fokusirali na to da implementiramo dio po dio te u konačnici te dijelove jednostavno spojimo, za svaku sekciju zahtjeva ćemo implementirati funkciju za parsiranje.

Započmimo s parsiranjem zahtjevne linije.

3.2.1 *Request-Line*

Request-Line se sastoji od sljedećih komponenti: Tokena za metodu, URI zahtjeva i verzija protokola. Svi ovi elementi su razdvojeni oznakom za razmak SP. Na kraju linije ide CRLF.

Primjer 3.2: Request-Line

```
1 Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Token za metodu, odnosno tražena metoda, može biti jedna od: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT.

Budući da želimo imati GET metodu podržanu, stvari su relativno jednostavne. Možemo započeti tako da string zahtjeva rastavimo po znaku za novi red. Nazovimo ovu metodu SplitRequest. Metoda će primiti string zahtjeva i vratiti niz linija gdje je svaka linije jedan red zahtjeva. Za dobivanje niza linija možete koristiti metodu `Split()` te nemojte zaboraviti da je potrebno ukloniti početne i krajnje razmake (metoda `Trim()`).

Metoda u konačnici može izgledati ovako:

Primjer 3.3: SplitRequest metoda

```
1        private string[] SplitRequest(string request)
2        {
3        ...
4        }
```

Nastavljamo sa sljedećim parserom: parser za *Request-Line* - ParseRequestLine. Podsjetite se od čega se ta linija sastoji. Za početak treba liniju rastaviti po razmaku budući da je razmak oznaka kojom su razdvojeni elementi linije. Ovaj parser kao povratnu vrijednost mora vratiti metodu koju je klijent zatražio (GET, POST...), URI i verziju protokola. Sve ove elemente možemo vratiti kao uređeno trojku koristeći strukturu `Tuple` koja postoji u C#-u ili izradom svoga objekta koji će sadražvati reference na ove informacije (sami odaberite). U slučaju da nije primljena metoda GET u zahtjevu, potrebno je vratiti vrijednost koja će ukazati na nepodržanu metodu. Metodu GET i verziju HTTP-a koja se koristi prikažite enumima (dodajte u oba enuma vrijednost Unsupported za korištenje ako je dobivena neočekivana vrijednost).

Primjer 3.4: ParseRequestLine metoda

```
1        private Tuple<Method, string, Version>
             ParseRequestLine(string requestLine)
2        {
3        ...
4        }
```

Probamo li ovu metodu vidjet ćemo da smo uspješno parsirali prvu liniju zahtjeva.

3.2.2 *Headers*

HTTP zaglavlja pružaju potrebne informacije o zahtjevu ili odgovoru na zahtjev, ili o objektu koji se šalje putem tijela poruke (engl. *message-body*). Četiri su tipa zaglavlja:

- *general-header* - u ovo zaglavlje idu polja koja su primjenjiva i kod zahtjeva i kod odgovora;
- *request-header* - ova zaglavlja su primjenjiva kod zahtjeva;
- *response-header* - ova zaglavlja su primjenjiva kod odgovora;
- *entity-header* - zaglavlja u koja se spremaju podatci o tijelu poruke ili o resursu koji je identificiran zahtjevom. U ova zaglavlja spadaju infomracije o duljini i tipu poruke na primjer.

Sva zaglavlja imaju siti format, a sva polja se sastoje od imena koje završava s dvotočkom nakon koje slijedi vrijednost pripadajućeg polja. Ova zaglavlja skupa s vrijednostima ćemo spremiti u jednu listu. Dohvatite sve linije zahtjeva, izuzev prvu, do prazne linije. Dobivene linije pošaljite u metodu za prsiranje zaglavlja - ParseHeaders. Metoda će kao rezutlat vratiti listu uređenih parova - polje-vrijednost. Uočite da po specifikaciji postoji prazan red između zaglavlja i tijela poruke (*message-body*) - iz ovog razloga uzimamo linije dok ne dođemo do praznog reda.

Primjer 3.5: ParseHeaders metoda

```
1        private List<Tuple<string, string>>
             ParseHeaders(string[] lines)
2        {
3        ...
4        }
```

3.2.3 *Message-body*

Posljednji dio parsera koji nam je potreban je onaj za parsiranje tijela poruke. Kada bismo išli raditi stvarni parser za tijelo morali bismo izvlačiti informacije iz zaglavlja o tipu i duljini poruke i slično. Ovdje ćemo zbog jednostavnosti pretpostaviti da je tijelo poruke sve ono što nije u *request-line* i zaglavljima. Tijelo poruke ćemo vratiti kao jedan string (ne kao niz kako bi vam vjerojatno bilo u ovome trenutku ako ste pratili ovaj tekst). S obzirom na to da samo spajamo stringove, ne morate izvlačiti u posebnu metodu (iako je preporučljivo).

Primjer 3.6: ParseMessageBody

```
1         private string ParseMessageBody(string[] messageBody)
2         {
3         ...
4         }
```

Za dovršiti parser potrebno je napraviti metodu koja će pozvati ove već navedene. Neka se nova metoda zove GetRequestInfo i neka prima string zahtjeva i vrati RequestInfo objekt.

Implementacija RequestInfo objekta:

Primjer 3.7: RequestInfo klasa

```
1  class RequestInfo
2  {
3          public Tuple<Method, string, Version> RequestLine { get;
               private set; }
4          public List<Tuple<string, string>> Headers { get;
               private set; }
5          public string MessageBody { get; private set; }
6
7      public RequestInfo(Tuple<Method, string, Version>
           requestLine, List<Tuple<string, string>> headers, string
           mbody)
8      {
9          RequestLine = requestLine;
10         Headers = headers;
11         MessageBody = mbody;
12     }
13
14     public override string ToString()
15     {
16         string requestLine = "( " + RequestLine.Item1 + ", " +
               RequestLine.Item2 + ", " + RequestLine.Item3 + "
               )\n";
17         string headers = Headers.Select(x => x.Item1 + " : " +
               x.Item2 + "\n").Aggregate((x, y) => x + y);
18
19         return requestLine + headers + "\n" + MessageBody + "\n";
20     }
21 }
```

Primjer 3.8: GetRequestInfo metoda

```
1         public RequestInfo GetRequestInfo(string request)
```

```
        {
        ...
        }
```

Ovime smo gotovi s prvim dijelom HTTP servera te možemo nastaviti dalje. U ovome trenutku možete testirati server na način da u metodu za parsiranje pošaljete primjer zahtjeva kao što je dan u primjeru 3.9.

Primjer 3.9: Primjer zahtjeva

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

3.3 Implementacija TCPListenera i primanje zahtjeva

Sada kada smo napokon dovršili parser, potrebno je pokrenuti TCPListener i uz pomoć njega primiti zahtjev klijenta. Klijent za ovu vježbu će nam biti web browser. Zahtjevamo da server može posluživati više klijenata. Ovo je nešto što smo radili na prethodnim vježbama te iste primjere možete iskoristiti sada. Najveća izmjena koju je potrebno napraviti jest da se ne odžava konekcija sa browserom. Naime, kada bi održavali konekciju, browser bi vječito čekao dok se konekcija zatvori. Možete ovo pokušati. Ono što je također jako važno jest da StreamReader ReadLineAsync metoda koju koristimo čita iz streama sve liniju po liniju te je potrebno modificirati taj dio koda da se pročita cijeli string. Po specifikaciji RFC2616 jedna prazna linija stoji između zaglavlja i tijela. Tijelo ćemo ignorirati. Ako ste sve napravili ispravno, otvaranjem web browsera te odlaskom na `localhost:broj_porta browser` će prikazati poruku koju pošaljete (u primjeru s prošlih vježbi "200 OK").

Primjer 3.10: Primanje poruke dok god ima znakova za pročitat

```
string received = "";
string receivedBuffer = "";

do
{
    received = await reader.ReadLineAsync();
    receivedBuffer = receivedBuffer + received + "\n";
} while (received != "");
```

Uočite da u kodu poviše ne čitamo iz NetworkStreama, već iz MemoryStreama.

3.3.1 Slanje datoteka

Za uspješno slanje datoteke potrebno je odrediti nekoliko stvari:

1. Odrediti postoji li datoteka. U slučaju da ne postoji poslati HTML stranicu "Notfound.html" koja sadrži html elemente dovoljne za prikaz teksta "404 Not found";
2. Odrediti ekstenziju datoteke (Path.GetExtension(file)), neovisno o tome koja datoteka bila. Podržavamo dvije ekstenzije: jpeg i html;
3. Na osnovu ekstenzije odrediti Content-Type: "text/html" ili "image/jpeg";

4. Učitati bajtove datoteke - File.GetAllBytes(file). U slučaju da nije specificirano koja se datoteka traži, tj. da je URL izvučen iz zahtjeva jednostavno "/", onda se pretpostavlja da klijent želi stranicu `index.html`;
5. Složiti odgovor koji ćemo poslati browseru te ubaciti informacije o response kodu, Content-Type-u i Content-Length-u. Primjer za predložak je ispod;
6. ResponseCode koji vraćate može biti "200 OK", "404 Not found", "500 Internal server error". Prvi vraćate ako je sve ok, drugi ako je korisnik zatražio dokument koji ne postoji, a posljednji ako se dogodi kakva greška (exception);
7. Zapisati u stream prvo zaglavlje (skupa sa jednim praznim redom na kraju - specifikacija), a zatim sadržaj datoteke. Dakle, možete zaključiti da sadrđaj datoteke ide u message-body dio HTTP odgovora.

Moramo imati na umu da browser kada šalje URL u request-line-u prdstavlja relativnu putanju. Dakle, sadržaj može biti npr. "/index.html" i mi onda moramo pronaći gdje se "index.html" nalazi te postupiti po koracima poviše.

Za početak ćemo napraviti da radi za jednu html datoteku, a kasnije nadograditi da se ovisno o tome je li pronađena datoteka i je li se dogodila pogreška na serveru, isti vrati html stranicu s odgovarajućom porukom (404 ili 500).

Kada sve ovo proradi, dodajemo mougćnost za određivanje Content-Type-ova ovisno o tome je li ekstenzija datoteke html ili jpg, a za ostale možemo postaviti Content-Type na `application/octet-stream`.

3.4 Zadatci

Cijela ova vježba je jedan veliki zadatak, ali ga možemo rastaviti u nekoliko koraka:

1. Implementirajte HTTP parser koji zadovoljava specifikacije kako je dogovoreno u ovome dokumentu. Mora sadržavati navedene metode s potpisima, a bilo kakve dodatne metode su opcionalne.
2. Nakon što ste uspješno implementirali i testirali HTTP parser da uistinu radi, potrebno je implementirati server koji može posluživati više klijenata (async-await), ali koji neće održavati konekciju. Uzmite i modificirajte MultipleClientExample s prethodnih vježbi.
3. StreamReader metoda ReadLineAsync čita jednu po jednu liniju, stoga je potrebno ili koristiti drugu metodu ili čitati liniju po liniju dok se ne dođe do prazne linije. Nakon čega ide parsiranje.
4. Za početak vratite rezultat "200 OK" da se uvjerite da vam program ispravno radi - ispis bi trebali viditi u web browseru.
5. Po završetku prethodnog zadatka, dodajte mogućnost slanja html i jpg datoteka web browseru. Detalje imate opisane poviše u 7 točaka. Datoteke koje ćete slati browseru možete držati na prozivoljnoj lokaciji - preporuka je unutar nekog foldera u projektu.
6. Ako ste sve ispravno implementirali, odlaskom na localhost:<port> bi ste trebali vidjeti "Index" stranicu, odlaskom na localhost:<port>/<nepostojeći_sadržaj> bi ste trebali vidjeti "Not found" stranicu. Odlaskom na localhost:<port>/debian.jpg bi trebali vidjeti sliku.
7. U slučaju da je primljena metoda različita od GET, vratite "500 Internal server error".

Ostali zadatci:

1. Izmijenite klijentsku windows forms aplikaciju s prošlih vježbi na način da možete poslati sliku serveru. Ne morate realizirati kao HTTP zahtjev, dapače, stvaranje POST zahtjeva i njegova obrada kojom biste ovo realizirali su prekompleksni za jedne vježbe. U slučaju da želite, slobodno, ali je potrebno proučiti RFC 2616 specifikaciju detaljno.