
Aplikacijski sloj: Web, HTTP i REST API

Poglavlje 2 — Skripta kolegija *Računalne mreže*

doc. dr. sc. Sandi Baressi Šegota

Fakultet informatike Sveučilišta Jurja Dobrile u Puli

Poglavlje 2

Aplikacijski sloj: Web, HTTP i REST API

Aplikacijski je sloj onaj koji korisnik zapravo „vidi”: u njemu žive web, e-pošta, streaming i sve ostale mrežne usluge. Za razliku od nižih slojeva, aplikacije se izvode isključivo na *krajnjim uređajima* — mreža ih samo povezuje, pa programer mrežne aplikacije ne mora pisati softver za rutere. U ovom poglavlju upoznajemo načela po kojima se grade mrežne aplikacije, zatim detaljno protokol weba HTTP, te REST — danas najrašireniji stil za izgradnju web servisa nad HTTP-om.

2.1 Načela mrežnih aplikacija

2.1.1 Arhitekture: klijent–poslužitelj i P2P

Mrežne aplikacije obično slijede jednu od dviju arhitektura. U arhitekturi **klijent–poslužitelj** postoji stalno dostupan *poslužitelj* s poznatom, nepromjenjivom adresom (najčešće u podatkovnom centru) kojemu se povremeno spojeni *klijenti* obraćaju sa zahtjevima; klijenti međusobno ne komuniciraju izravno. U **P2P** (*peer-to-peer*) arhitekturi nema stalnog poslužitelja, nego ravnopravni čvorovi (*peerovi*) izravno razmjenjuju usluge. P2P se dobro skalira jer svaki novi peer donosi i vlastite resurse, no upravljanje je složenije zbog stalne promjenjivosti čvorova.



Figure 2.1: Dvije temeljne arhitekture mrežnih aplikacija.

2.1.2 Procesi, socketi i adresiranje

Komunikaciju zapravo obavljaju *procesi* koji se izvode na hostovima. Proces šalje i prima poruke kroz **socket** — programsko sučelje (svojevrsna „vrata”) između aplikacije i transportnog sloja. Da bi primio poruku, proces mora biti jednoznačno adresiran, i to *dvjema* informacijama: **IP adresom** koja identificira host i **brojem porta** koji identificira konkretan proces (uslugu) na tom hostu. Tako HTTP poslužitelj tipično sluša na portu 80 (odnosno 443 za HTTPS), SMTP na 25, a DNS na 53.

Socket

Socket je apstrakcija koja aplikaciji omogućuje slanje i primanje podataka bez poznavanja detalja prijenosa. Aplikacija „ubaci“ poruku u socket, a transportni sloj brine o njezinoj dostavi drugom socketu.

2.1.3 Zahtjevi aplikacija i transportne usluge

Različite aplikacije imaju različite potrebe: prijenos datoteke zahtijeva *potpunu pouzdanost*, dok streaming tolerira poneki izgubljeni paket ali traži dostatnu *propusnost*; mrežne igre i pozivi osjetljivi su na *kašnjenje*. Internet nudi dva glavna transportna protokola. **TCP** pruža pouzdan prijenos, kontrolu toka i zagušenja, uz cijenu uspostave veze i većih troškova. **UDP** je jednostavan i brz, bez jamstva dostave i bez kontrole. Važno je uočiti da *nijedan* od njih ne jamči propusnost ni najveće kašnjenje — te se potrebe rješavaju na višim razinama (npr. prilagodljivim streamingom).

Aplikacija	Protokol	Transport
web	HTTP	TCP
e-pošta	SMTP, IMAP	TCP
prijenos datoteka	FTP	TCP
imenički sustav	DNS	UDP
streaming	HTTP/DASH	TCP
IP telefonija	SIP, RTP	UDP

Table 2.1: Tipičan izbor transportnog protokola po aplikaciji.

2.2 Web i HTTP

2.2.1 Osnovni pojmovi i pregled HTTP-a

Web stranica sastoji se od *objekata* (HTML datoteka, slike, CSS, skripte, video), a svaki je objekt adresiran **URL-om** oblika `protokol://host[:port]/staza`. Osnovna HTML datoteka sadrži reference na ostale objekte koje preglednik potom dohvaća.

Protokol weba je **HTTP** (*HyperText Transfer Protocol*). Radi po modelu *zahtjev–odgovor*: klijent (preglednik) šalje zahtjev, poslužitelj vraća odgovor. HTTP koristi TCP i **bez je stanja** (*stateless*) — poslužitelj ne pamti prethodne zahtjeve, što pojednostavljuje izvedbu i olakšava skaliranje.

2.2.2 Nepersistentni i persistentni HTTP

U **nepersistentnom** HTTP-u za svaki se objekt otvara nova TCP veza, pa dohvat svakog objekta stoji približno *dva vremena povratnog puta* (RTT) — jedan RTT za uspostavu veze i jedan za zahtjev i početak odgovora — uvećano za vrijeme prijenosa. U **persistentnom** HTTP-u (zadano od HTTP/1.1) jedna se TCP veza koristi za više objekata, a zahtjevi mogu ići i *u nizu* (pipelining), čime se broj RTT-ova bitno smanjuje.

Riješeni primjer

Za stranicu s 1 HTML datotekom i 5 slika (6 objekata) i $RTT = 50$ ms: nepersistentni pristup traži $\approx 2 \cdot 6 = 12$ RTT = 600 ms, dok persistentni s cjevovodom traži ≈ 3 RTT = 150 ms (uspostava + HTML, pa svih pet slika zajedno; zanemarujući vrijeme prijenosa).

2.2.3 Format poruke, metode i statusni kodovi

HTTP poruka ima *početni redak*, niz *zaglavlja* (svako u svom retku), prazan redak, te neobavezno *tijelo*. Zahtjev počinje metodom i stazom (npr. GET /index.html HTTP/1.1), a odgovor statusnim retkom (npr. HTTP/1.1 200 OK).



Figure 2.2: Struktura HTTP zahtjeva i odgovora.

Najvažnije **metode** su GET (dohvat), POST (slanje podataka u tijelu), HEAD (samo zaglavlja), PUT (postavljanje/zamjena) i DELETE (brisanje). Razlika između GET-a i POST-a je važna: GET nosi parametre u URL-u (vidljiv je, može se označiti i predmemorirati, te je siguran i idempotentan), dok POST šalje podatke u tijelu i prikladan je za veće ili osjetljive podatke te za radnje koje mijenjaju stanje. **Statusni kodovi** grupirani su po prvoj znamenki: 2xx uspjeh (200 OK), 3xx preusmjeravanje (301, 304 Not Modified), 4xx pogreška klijenta (404 Not Found) i 5xx pogreška poslužitelja (500).

2.2.4 Kolačići, predmemorija i uvjetni GET

Budući da je HTTP bez stanja, za pamćenje konteksta (prijava, košarica) koriste se **kolačići**. Poslužitelj u odgovoru pošalje zaglavlje Set-Cookie s identifikatorom; klijent ga pohrani i vraća pri svakom idućem zahtjevu zaglavljem Cookie, pa ga poslužitelj prepoznaje.

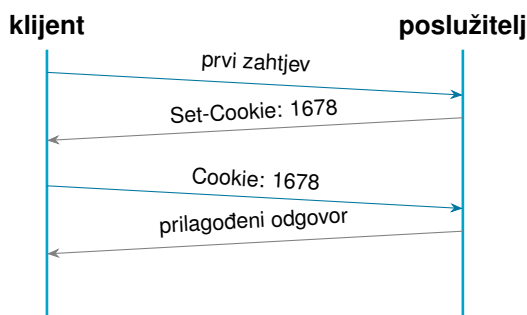


Figure 2.3: Kolačići dodaju stanje nad protokolom bez stanja.

Web predmemorija (proxy) čuva kopije objekata bliže korisniku: pri *pogotku* (hit) odgovara odmah, a tek pri *promašaju* (miss) dohvaća objekt s izvornog poslužitelja, čime se smanjuju kašnjenje i promet. Da bi predmemorirana kopija ostala svježa, koristi se **uvjetni GET**: klijent u zahtjev doda If-Modified-Since, a poslužitelj, ako objekt nije mijenjan, odgovara sa 304 Not Modified *bez tijela* — štedeći propusnost.

2.2.5 HTTPS i evolucija HTTP-a

Gotovo sav današnji web promet odvija se preko **HTTPS-a** — HTTP-a preko sloja TLS (port 443). TLS osigurava *povjerljivost* (enkripciju), *integritet* poruka i *autentikaciju* poslužitelja pomoću certifikata, pa preglednici obični HTTP danas označavaju kao nesiguran.

Sam se protokol razvijao radi smanjenja kašnjenja. **HTTP/1.1** uveo je persistentne veze, ali pati od „čekanja u redu” na razini veze. **HTTP/2** donosi binarni format i *multipleksiranje* više

tokova po jednoj vezi te kompresiju zaglavlja. **HTTP/3** radi nad protokolom QUIC (koji koristi UDP), čime ostvaruje bržu uspostavu veze i bolje ponašanje pri gubicima paketa.

2.3 REST API

2.3.1 Načela

REST (*Representational State Transfer*) arhitekturni je stil za izgradnju web servisa *nad* HTTP-om. Njegova su temeljna načela: sve je **resurs** identificiran **URI**-jem; nad resursom se djeluje **HTTP metodama** koje preslikavaju **CRUD** operacije (GET = čitaj, POST = stvori, PUT/PATCH = ažuriraj, DELETE = obriši); poslužitelj vraća **reprezentaciju** resursa, danas najčešće u formatu **JSON**. Komunikacija je **bez stanja** — svaki zahtjev sadrži sve što poslužitelju treba (npr. token u zaglavlju *Authorization*), pa bilo koji poslužitelj u nakupini može obraditi bilo koji zahtjev, što olakšava skaliranje.

Sigurne i idempotentne metode

Sigurna metoda ne mijenja stanje poslužitelja (GET, HEAD). *Idempotentna* metoda pri ponavljanju daje isti učinak (GET, PUT, DELETE). POST nije idempotentan — dvostruko slanje stvara dva zapisa. Zato su pri ponovnom slanju (npr. nakon izgubljenog odgovora) idempotentne metode sigurne.

2.3.2 Primjer

Dohvat pojedinog studenta metodom GET `/studenti/42` vraća 200 OK s tijelom:

```
{ "id": 42, "ime": "Ana", "godina": 2, "aktivan": true }
```

Stvaranje novog studenta metodom POST `/studenti` (s podacima u tijelu) vraća 201 Created i zaglavlje *Location* s URI-jem novog resursa. Uspješan DELETE obično vraća 204 No Content.

2.3.3 Dobra praksa

Preporučuje se: koristiti *imenice* (ne glagole) i množinu za kolekcije (`/studenti/42/ocjene`); *verzionirati* API (`/api/v1/...`); dosljedno rabiti statusne kodove i ujednačen format grešaka; te podržati stranicanje i filtriranje (`?stranica=2&limit=20`).

2.3.4 Autentikacija

Budući da je REST bez stanja, identitet se ne pamti u sesiji na poslužitelju, nego klijent uz *svaki* zahtjev prilaže vjerodajnice, tipično u zaglavlju *Authorization: Bearer <token>*. Token se prethodno dobiva prijavom (npr. standardima *OAuth 2.0* / *JWT*), a poslužitelj ga provjerava pri svakom pozivu. Time se zadržava statelessness i omogućuje da bilo koji poslužitelj u nakupini obradi zahtjev, uz jasno odvajanje autentikacije od poslovne logike.

2.3.5 REST u odnosu na druge stilove

Iako je REST danas najrašireniji, nije jedini pristup. *RPC* (*remote procedure call*, npr. gRPC) modelira komunikaciju kao pozive udaljenih funkcija i često je vrlo učinkovit uz strogo definiran ugovor. Stariji *SOAP* koristi XML i vrlo formalna pravila te se još susreće u nekim poslovnim sustavima. Noviji *GraphQL* omogućuje klijentu da jednim upitom zatraži točno

željena polja, čime izbjegava dohvat suvišnih podataka. Izbor stila ovisi o potrebama: REST je odličan zadani izbor za javna, jednoobrazna web sučelja.

Sažetak poglavlja

Mrežne aplikacije grade se kao klijent–poslužitelj ili P2P; procesi komuniciraju kroz *sockete* adresirane IP adresom i portom. **HTTP** je protokol weba: model zahtjev–odgovor nad TCP-om, bez stanja, s persistentnim vezama, kolačićima, predmemorijom i uvjetnim GET-om za učinkovitost. **REST** nad HTTP-om nudi jednoobrazno sučelje: resursi (URI), metode (CRUD), JSON reprezentacije i komunikacija bez stanja.