

Aplikacijski sloj I: Web, HTTP i REST API

Predavanje 2 • kolegij Računalne mreže

doc. dr. sc. Sandi Baressi Šegota

Fakultet informatike Sveučilišta Jurja Dobrile u Puli

Sadržaj predavanja

- 1 Načela mrežnih aplikacija
- 2 Web i HTTP
- 3 REST API
- 4 Sažetak

Načela mrežnih aplikacija

Ishodi učenja

Nakon ovog predavanja moći ćete:

- razlikovati arhitekture klijent–poslužitelj i P2P;
- objasniti ulogu procesa, socketa i adresiranja (IP + port);
- opisati rad HTTP-a: (ne)persistentnost, poruke, metode, statusne kodove;
- objasniti kolačiće, web predmemoriju i uvjetni GET;
- projektirati jednostavan **REST API** (resursi, metode, JSON).

Mrežne aplikacije oko nas

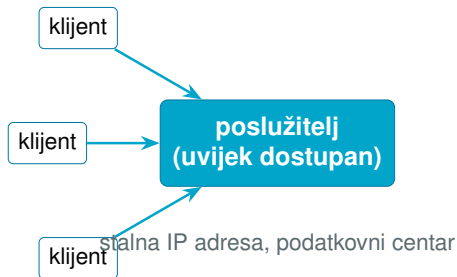
- web i pretraživanje, e-pošta, poruke
- društvene mreže, streaming videa i glazbe
- mrežne igre, videokonferencije
- aplikacije u oblaku (SaaS), IoT

Ključna ideja

Aplikacije se izvode **na rubu** (hostovima); mreža ih samo povezuje. Ne treba nam pisati softver za rutere.

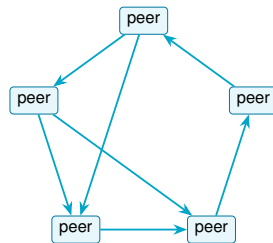
Arhitektura: klijent–poslužitelj

- **Poslužitelj**: uvijek dostupan, stalna adresa, u podatkovnom centru
- **Klijenti**: povremeno spojeni, promjenjive adrese, ne komuniciraju izravno



Arhitektura: P2P (ravnopravni čvorovi)

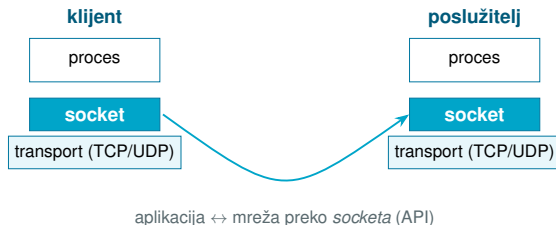
- nema stalnog poslužitelja
- **peerovi** izravno razmjenjuju usluge
- dobra **skalabilnost** (novi peer donosi i kapacitet)
- složeno upravljanje (promjenjivost)



nema stalnog poslužitelja; peerovi izravno razmjenjuju

Procesi i socketi

- aplikacija = **procesi** koji komuniciraju
- **socket** = vrata između procesa i transporta
- proces šalje/prima poruke kroz socket



Adresiranje procesa

- za primanje poruka proces treba **identifikator**
- identifikator = **IP adresa** (host) + **broj porta** (proces)
- primjeri portova: 80 (HTTP), 443 (HTTPS), 25 (SMTP), 53 (DNS)

Analogija

IP adresa je „zgrada”, a broj porta „stan” u kojem živi konkretna usluga.

Što aplikacija treba od transporta?

- **Pouzdanost**: smiju li se podaci gubiti? (prijenos datoteka: ne; audio: donekle da)
- **Propusnost**: treba li zajamčenu brzinu? (streaming da; e-pošta ne)
- **Kašnjenje**: koliko je osjetljiva? (igre, pozivi: vrlo)
- **Sigurnost**: enkripcija, integritet, autentikacija

Transportne usluge: TCP vs UDP

TCP	UDP
pouzdan prijenos	bez jamstva dostave
kontrola toka i zagušenja	bez kontrole
uspostava veze (handshake)	bez uspostave
veći troškovi, sporiji start	mali troškovi, brz
web, e-pošta, prijenos datoteka	streaming, DNS, igre

Ni TCP ni UDP ne jamče propusnost ni kašnjenje — to rješavaju viši mehanizmi.

Aplikacije i njihov transport

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

Klijent–poslužitelj vs P2P

Klijent–poslužitelj	P2P
stalni, adresabilni poslužitelj	nema stalnog poslužitelja
jednostavno upravljanje	složena koordinacija
skalira uz više resursa	skalira s brojem peerova
web, e-pošta, cloud	dijeljenje datoteka, blockchain

Web i HTTP

Web: osnovni pojmovi

- **Web stranica** sastoji se od **objekata** (HTML, slike, CSS, JS, video)
- svaki objekt adresiran je **URL-om**
- `http://www.fipu.hr/kolegiji/mreze.html`
- HTML datoteka sadrži reference na ostale objekte

URL

protokol `://` host `[: port]` `/` staza

HTTP: pregled

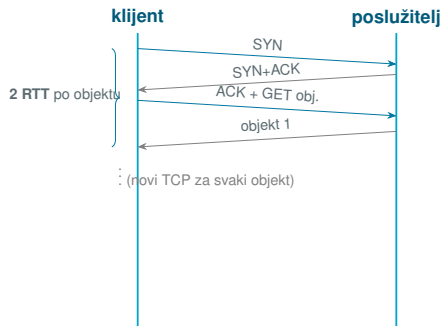
- **HyperText Transfer Protocol** — protokol weba, model **zahtjev–odgovor**
- koristi **TCP** (port 80; HTTPS: TLS + 443)
- **bez stanja** (stateless): poslužitelj ne pamti prethodne zahtjeve

Zašto bez stanja?

Jednostavnost i skalabilnost. Stanje (npr. prijava) dodaje se **kolačićima**.

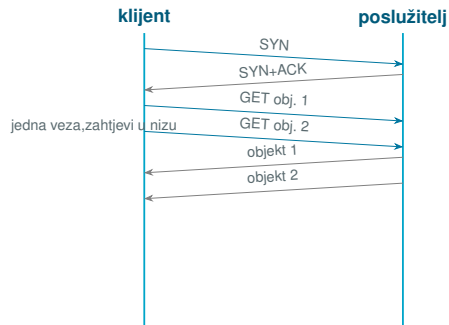
Nepersistentni HTTP

- **nova TCP veza** za svaki objekt
- po objektu: **2 RTT** + vrijeme prijenosa
- mnogo objekata $\Rightarrow$ mnogo veza



Persistentni HTTP

- **jedna** TCP veza za više objekata
- zahtjevi mogu ići **u nizu** (pipelining)
- manje RTT-ova, brže učitavanje
- zadano od HTTP/1.1



Riješeni primjer: vrijeme učitavanja

Zadano

- 1 HTML + 5 slika = 6 objekata
- RTT = 50 ms

Nepersistentni: $\approx 2 \text{ RTT} \times 6 = 12 \text{ RTT} = 600 \text{ ms}$

Persistentni (pipelining)

$\approx 2 \text{ RTT}$ (veza + HTML) + 1 RTT (svih 5 slika)
 $= 3 \text{ RTT} = 150 \text{ ms}$

(uz zanemarivanje vremena prijenosa objekata)

Format HTTP poruke

zahtjev

```
GET /index.html HTTP/1.1
Host: www.fipu.hr
User-Agent: Mozilla/5.0
Accept: text/html
(prazan redak = kraj zaglavlja)
```

odgovor

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<html> ... </html>
```

Poruka: **početni redak** + **zaglavlja** (redak po redak) + prazan redak + neobavezno **tijelo**.

HTTP metode

- **GET** — dohvat objekta (parametri u URL-u)
- **POST** — slanje podataka u tijelu (npr. obrazac)
- **HEAD** — kao GET, ali samo zaglavlja (bez tijela)
- **PUT** — postavi/zamijeni objekt na zadanom URL-u
- **DELETE** — obriši objekt

Statusni kodovi

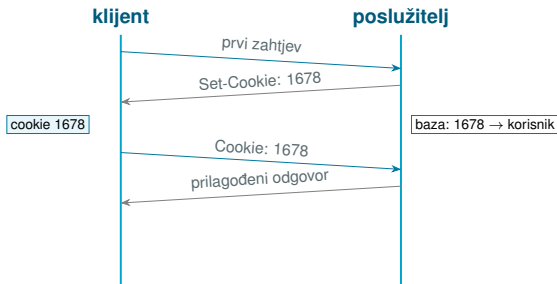
- **2xx** uspjeh — 200 OK, 201 Created
- **3xx** preusmjerenje — 301 Moved, 304 Not Modified
- **4xx** greška klijenta — 400 Bad Request, 404 Not Found
- **5xx** greška poslužitelja — 500 Internal Error, 503 Unavailable

Zapamtiti

Prva znamenka nosi **kategoriju** ishoda — pravi jezik između klijenta i poslužitelja.

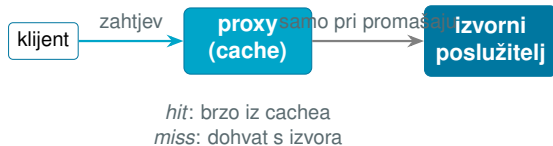
Kolačići (cookies): stanje nad HTTP-om

- 1 poslužitelj šalje Set-Cookie
- 2 klijent sprema kolačić
- 3 pri idućem zahtjevu šalje Cookie
- 4 poslužitelj prepoznaje korisnika



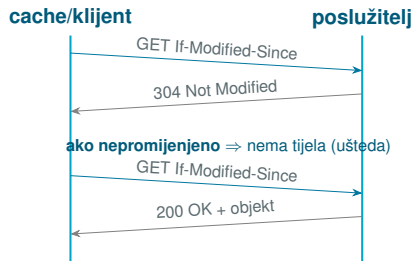
Web predmemorija (proxy cache)

- posrednik čuva kopije objekata
- **hit**: odgovor iz cachea (brzo)
- **miss**: dohvat s izvora
- smanjuje kašnjenje i promet



Uvjetni GET

- cache pita: je li objekt promijenjen?
- If-Modified-Since: <datum>
- ako nije: 304 Not Modified (bez tijela)
- štedi propusnost, čuva svježinu



GET vs POST

GET	POST
parametri u URL-u	podaci u tijelu
vidljiv, može se označiti/keširati	prikladan za veće/osjetljive podatke
siguran i idempotentan	mijenja stanje
dohvat resursa	slanje obrasca, stvaranje

Važna HTTP zaglavlja

- Host — ciljani poslužitelj (obavezno u HTTP/1.1)
- User-Agent — preglednik/klijent
- Content-Type — format tijela (`text/html`, `application/json`)
- Content-Length — veličina tijela
- Authorization — vjerodajnice (npr. token)
- Cache-Control — pravila predmemoriranja

HTTPS: siguran web

- HTTP preko **TLS-a** (port 443)
- pruža **povjerljivost** (enkripcija), **integritet** i **autentikaciju** poslužitelja
- certifikat potvrđuje identitet poslužitelja

Danas standard

Gotovo sav web promet je HTTPS; preglednici označavaju obični HTTP kao nesiguran.

Evolucija: HTTP/1.1, HTTP/2, HTTP/3

- **HTTP/1.1** — persistentne veze, tekstualne poruke; problem „čekanja u redu”
- **HTTP/2** — binarno, **multipleksiranje** više tokova po jednoj vezi, kompresija zaglavlja
- **HTTP/3** — radi nad **QUIC**-om (UDP), brža uspostava, bolji pri gubicima

Nit vodilja

Svaka generacija smanjuje **kašnjenje** i bolje koristi vezu.

Predmemorija na više razina

- **preglednik** — lokalna predmemorija na uređaju
- **proxy** — zajednička predmemorija organizacije/ISP-a
- **CDN** — mreža poslužitelja blizu korisnika (detaljno u 3. predavanju)

Učinak

Manje kašnjenje, manji promet prema izvoru, veća otpornost na opterećenje.

Kolačići: koristi i privatnost

Koristi

- prijava i sesije
- košarica, postavke

Zabrinutosti

- praćenje kroz stranice
- privatnost i pristanak

Napomena

Kolačići trećih strana i regulativa (npr. obavijesti o privatnosti) mijenjaju način praćenja na webu.

REST API

Što je REST?

- **REST** = Representational State Transfer — arhitekturni **stil** za web servise
- gradi se **nad HTTP-om**: koristi njegove metode i statusne kodove
- sve je **resurs** identificiran URI-jem; poslužitelj vraća **reprezentaciju** (najčešće JSON)

Zašto REST?

Jednostavno, jednoobrazno i skalabilno sučelje između klijenta (web/mobilna aplikacija) i poslužitelja.

Resursi i URI, metode i CRUD

`/studenti/42`

resurs identificiran URI-jem

GET

POST

PUT/PATCH

DELETE

dohvati

stvari

ažuriraj

obriši

metode preslikavaju CRUD operacije na resurs

GET `/studenti` (svi) • GET `/studenti/42` (jedan) • POST `/studenti` (novi)

Reprezentacija: JSON

- tekstualni, čitljiv format
- parovi **ključ: vrijednost**, nizovi, ugniježdivanje
- Content-Type: application/json

GET /studenti/42 → 200 OK

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

Bez stanja (statelessness)

- svaki zahtjev sadrži **sve** što poslužitelju treba
- poslužitelj **ne pamti** prethodne zahtjeve klijenta
- stanje sesije nosi klijent (npr. token u zaglavlju)

Prednost

Bilo koji poslužitelj u nakupini može obraditi bilo koji zahtjev $\Rightarrow$ laka **skalabilnost**.

Sigurnost i idempotentnost metoda

- **Sigurna** metoda ne mijenja stanje: GET, HEAD
- **Idempotentna** metoda daje isti učinak pri ponavljanju: GET, PUT, DELETE
- POST **nije** idempotentan (dvostruko slanje → dva zapisa)

Zašto je važno

Kod ponovnog slanja (npr. zbog gubitka odgovora) idempotentne metode su **sigurne za ponavljanje**.

Statusni kodovi u REST-u

- 200 OK — uspješan GET/PUT
- 201 Created — uspješan POST (novi resurs)
- 204 No Content — uspješan DELETE
- 400 Bad Request — neispravan zahtjev
- 401/403 — neautenticiran / zabranjen
- 404 Not Found — resurs ne postoji

Dobra praksa REST API-ja

- imenice, ne glagoli u putanjama: `/studenti/42/ocjene`
- množina za kolekcije, gnijezdi odnose
- **verzioniranje**: `/api/v1/...`
- dosljedni statusni kodovi i format grešaka
- stranicanje i filtriranje: `?stranica=2&limit=20`

REST u odnosu na druge stilove

- **REST** — resursi + HTTP metode; jednostavno, rašireno
- **RPC** — pozivi udaljenih „funkcija” (npr. gRPC); brzo, ugovorno
- **SOAP** — stariji, XML, strogo definiran; još u nekim sustavima
- **GraphQL** — klijent traži točno željena polja jednim upitom

Primjer: stvaranje resursa (POST)

Zahtjev

```
POST /studenti
Content-Type: application/json
{ "ime": "Marko",
  "godina": 1 }
```

Odgovor

```
HTTP/1.1 201 Created
Location: /studenti/57
{ "id": 57, "ime": "Marko",
  "godina": 1 }
```

Autentikacija REST API-ja

- klijent šalje vjerodajnice u zaglavlju: `Authorization: Bearer <token>`
- token se dobiva prijavom (npr. OAuth 2.0 / JWT)
- poslužitelj provjerava token pri **svakom** zahtjevu (bez stanja)

Zašto token, a ne sesija?

Uz statelessness, token nosi identitet u svakom zahtjevu, pa svaki poslužitelj može obraditi zahtjev.

Alati za rad s API-jem

- `curl` — slanje zahtjeva iz naredbenog retka
- **Postman** / **Insomnia** — grafički klijenti za testiranje
- razvojni alati preglednika (kartica **Network**)
- **OpenAPI/Swagger** — opis i dokumentacija API-ja

Primjer

```
curl -X GET https://api.fipu.hr/v1/studenti/42
```

Sažetak

Ključne poruke

- aplikacije žive na rubu; arhitekture su **klijent–poslužitelj** i **P2P**
- procesi komuniciraju kroz **sockete**, adresirani **IP + port**
- **HTTP** je model zahtjev–odgovor nad TCP-om, bez stanja
- persistentni HTTP, kolačići, predmemorija i uvjetni GET poboljšavaju rad
- **REST** gradi jednoobrazno sučelje nad HTTP-om (resursi, metode, JSON)

Sljedeće predavanje

Aplikacijski sloj II: DNS, e-pošta, P2P, streaming i socket programiranje.

Pitanja za provjeru

- Koja je razlika između nepersistentnog i persistentnog HTTP-a?
- Čemu služe kolačići ako je HTTP bez stanja?
- Kako uvjetni GET štedi propusnost?
- Koje su metode idempotentne i zašto je to važno?
- Kako biste projektirali REST API za kolegije i studente?

Hvala na pažnji!

Pitanja i rasprava

doc. dr. sc. Sandi Baressi Šegota • FIPU