Arhitektura web aplikacija

Predmet: Bezbednost Aplikacija

Predavač: dr Dušan Stefanović

Web aplikacije dizajn i arhitektura

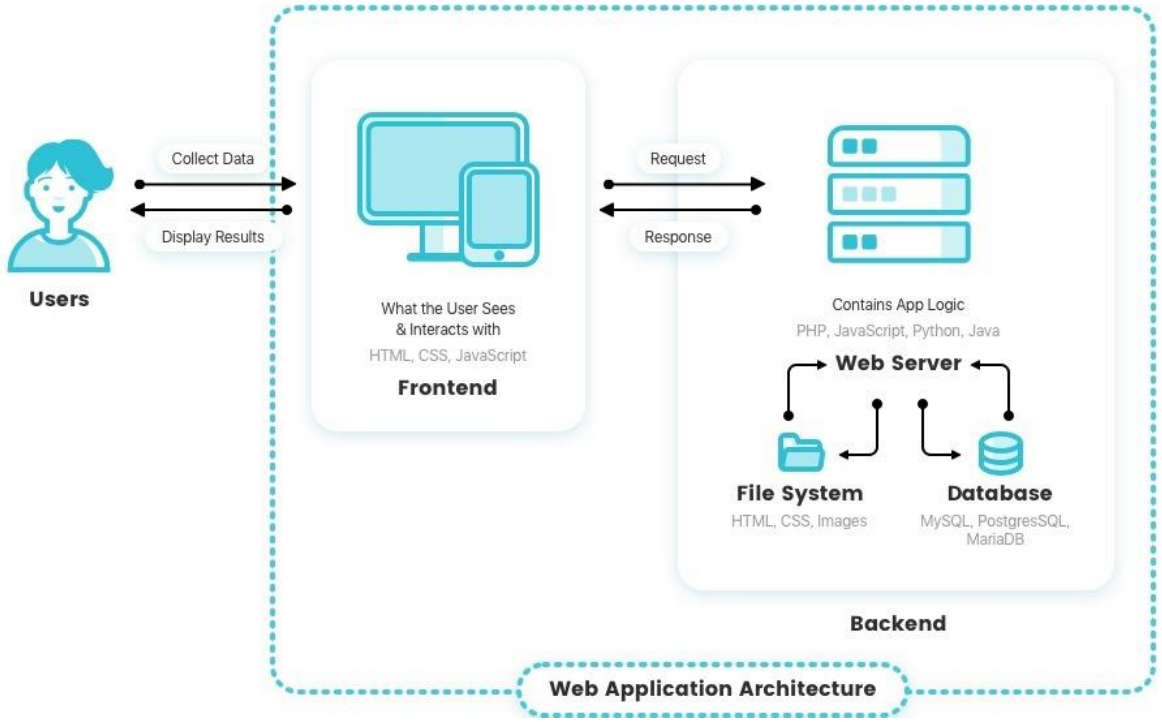
Razmena informacija od klijenta do servera i baze podataka

Dizajn HTTP protokola

Praćenje sesija pomoću kolačića (cookies)

HTML

Arhitektura Web aplikacije



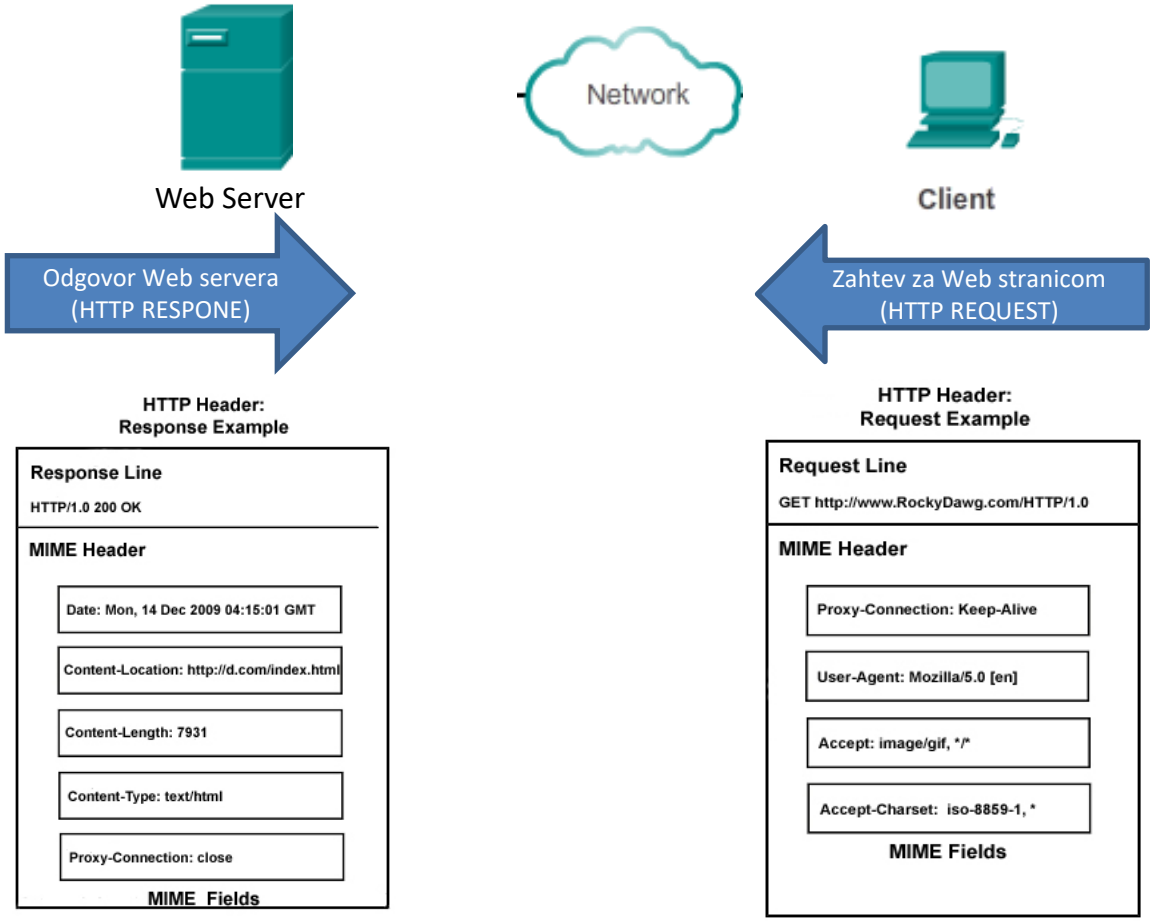
HTTP DIZAJN

HTTP je zadužen za komunikaciju između web servera i klijenta

HTTP je klijent server protokol u kojem klijent (web browser) kreira zahtev za server a web server vraća odgovor na zahtev (Request-Response protocol)

Odgovor servera je sadržaj u formi HTML stranica.

Prema standardnom podešavanju HTTP protokol koristi port 80



HTTP DIZAJN

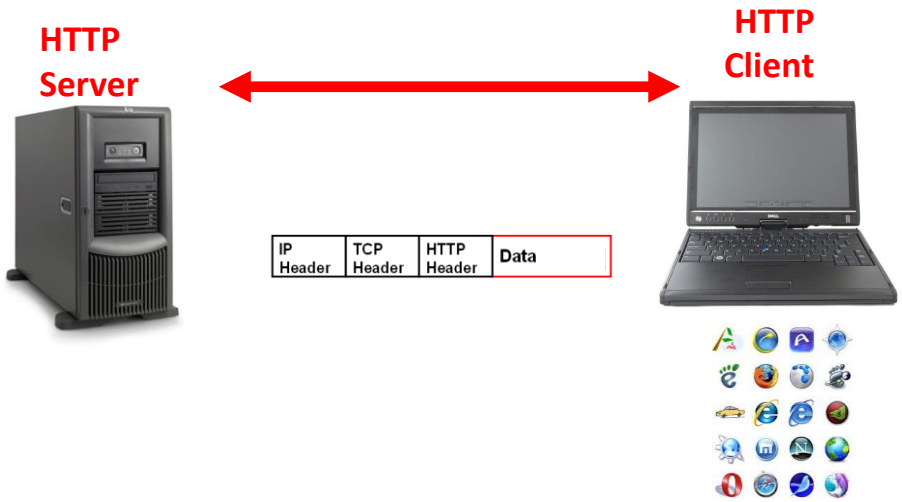
Najčešće upotrebljavana implementacija HTTP protokola je HTTP/1.1.

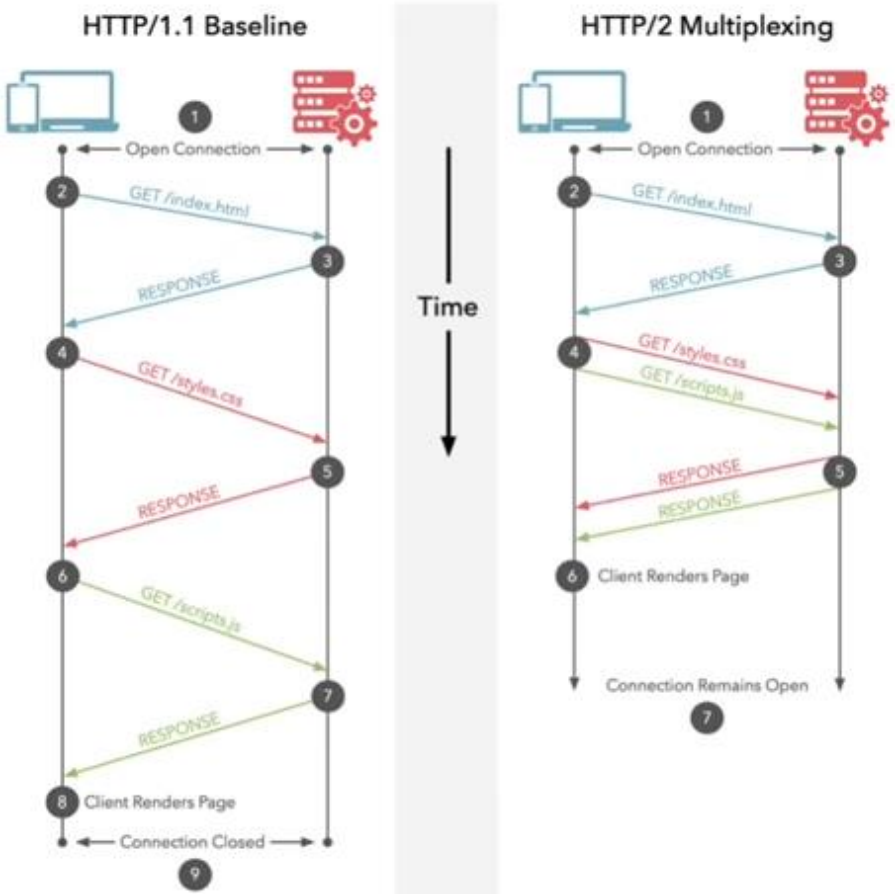
Najnovija verzija je HTTP/2

Fundamenti u odnosu na HTTP/1.1 su ostali isti

Osnovna razlika je u brzini tj. HTTP/2 je efikasnija u radu

Smanjeno je kašnjenje tako što je omogućeno potpuno request response multipexing





HTTP REQUEST PORUKA

```
GET /~srt/ HTTP/1.1
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET
   CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; InfoPath.1)
Host: www.vtsnis.edu.rs
Connection: Keep-Alive
```

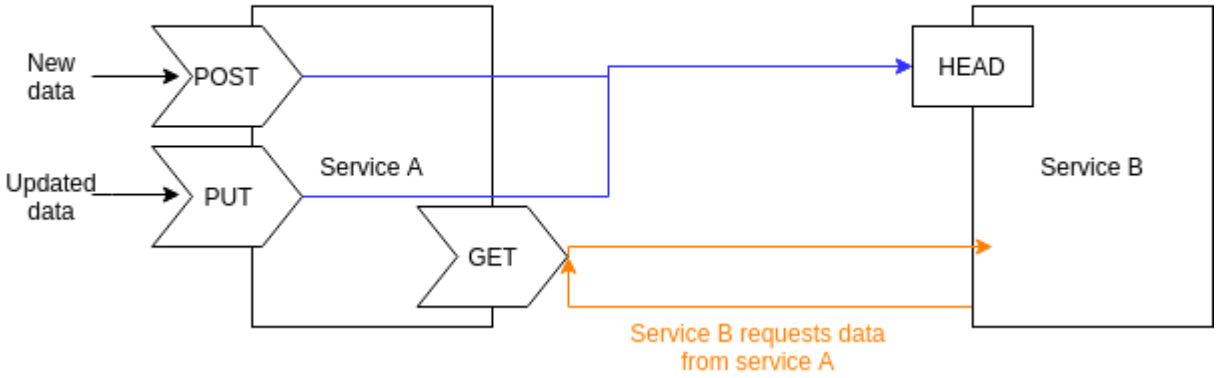
Request

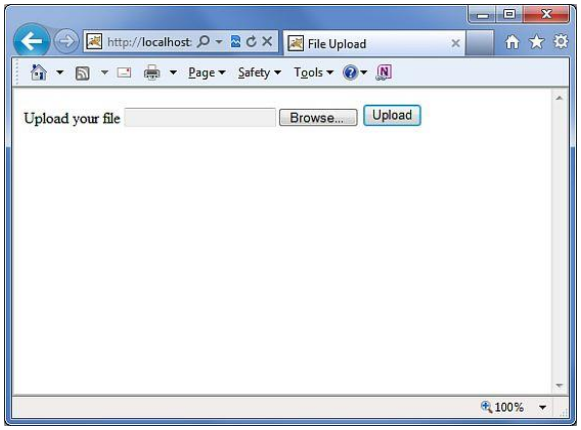
GET - Browser/klijent traži Web stranu

/~srt/ - Browser traži Web stranu u direktorijumu (index.html)

HTTP/1.1 – Browser koristi verziju HTTP/1.1

HTTP REQUEST KOMANDE





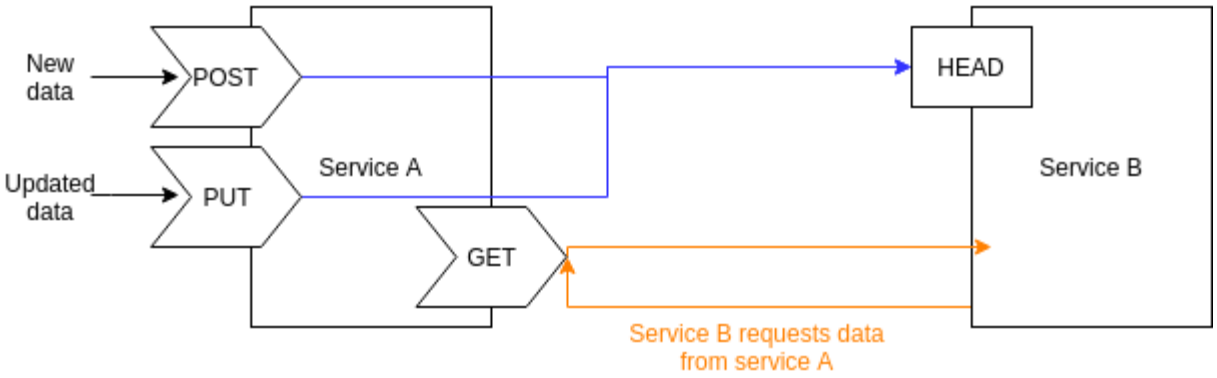
POST i **PUT** metode se koriste za slanje podataka (upload) na web server.

Kada korisnik unese podatak u formu koja je ugrađena u web strani, **POST** metodom šalje se podatak Web serveru.

PUT metodom se vrši ažuriranje podataka na web server, promena slike ili teksta na web sajtu.

DELETE metodom se brišu podaci na web server.

HTTP REQUEST KOMANDE

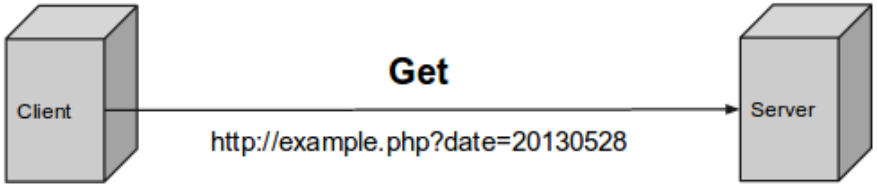


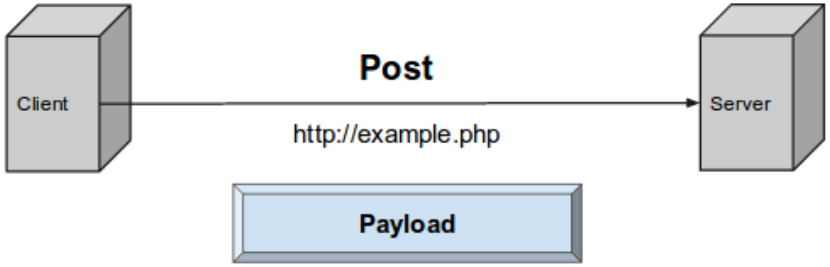
HEAD metoda identična GET metodi stim što server u odgovoru ne uključuje telo poruke.

TRACE metoda se koristi za prikaz posredničkih uređaja (proxy server, web firewall) koji se nalaze na putu do odredišnog web servera koji mogu da edituju HTTP zaglavlje i da se na taj način testira šta je primljeno na odredištu

OPTIONS metoda je upit web serveru za proveru dostupnih metoda za unetu URL adresu

HTTP REQUEST KOMANDE (GET vs POST)





GET metoda

Može da se iskoristi i za prosleđivanje podataka na web server.

Ne preporučuje se jer se ti podaci nalaze u URL adresi tj. lako su vidljivi ako se npr radi o korisničkom imenu ili lozinki.

Prihvatljiva je za slanje vrednosti koje treba primeniti za filtriranje podataka iz baze podataka.

POST metoda

Je način da se prosledi podatak preko forme na klijentskoj strani web serveru tj. bazi podataka radi provere autentifikacije ili upisa korisnika u bazu podataka.

Bezbedniji je način od get metode jer se vrednosti nalaze u zaglavlju HTTP poruke.

HTTP REQUEST ZAGLAVLJE

```
GET /~srt/ HTTP/1.1
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET
   CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; InfoPath.1)
Host: www.vtsnis.edu.rs
Connection: Keep-Alive
```

Header parametri

Accept-Language : - Korisnik zahteva dati jezik za traženi objekat

User-Agent: - Opisuje tip browser-a koji je poslao zahtev

Host: – Server na kome se nalazi traženi objekat

Connection: - Client/browser saopštava serveru da TCP konekciju drži otvorenom.

HTTP RESPONSE PORUKA


```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

HTTP RESPONSE PORUKA

```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Response poruka se sastoji:

- Statusna linija
- Header linija
- Sadržaj

HTTP RESPONSE PORUKA

```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Statusna Linija

`HTTP/1.1` – Server koristi HTTP/1.1 verziju

`200 OK` - Statusni kod, zahtev je uspešno obrađen i tražena informacija se nalazi u telu poruke

HTTP RESPONSE PORUKA

Statusni kod se nalazi u svakoj response poruci koju šalje server a koja opisuje status interakcije između klijenta i servera

HTTP statusni kodovi su podeljeni u 5 klasa:

1xx – Informacioni. Označava da je zahtev primljen ili da se obrađuje

2xx – Success. Označava da je zahtev isporučen i obrađen od strane servera

3xx – Preusmeravanje

4xx – Greška na strani klijenta. Server nema sve potrebne podatke koje je dobio od klijenta

5xx – Greška na strani servera .

Opšti statusni kodovi

200 OK

- Zahtev je uspešno obrađen i tražena informacija je u response poruci.

301 Moved Permanently

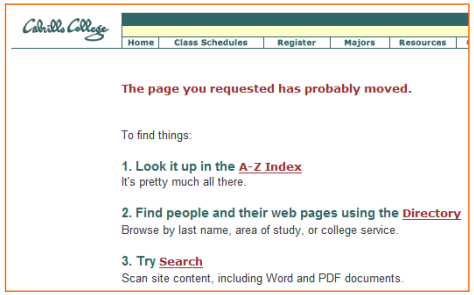
- Traženi objekat je trajno premešten.

400 Bad Request

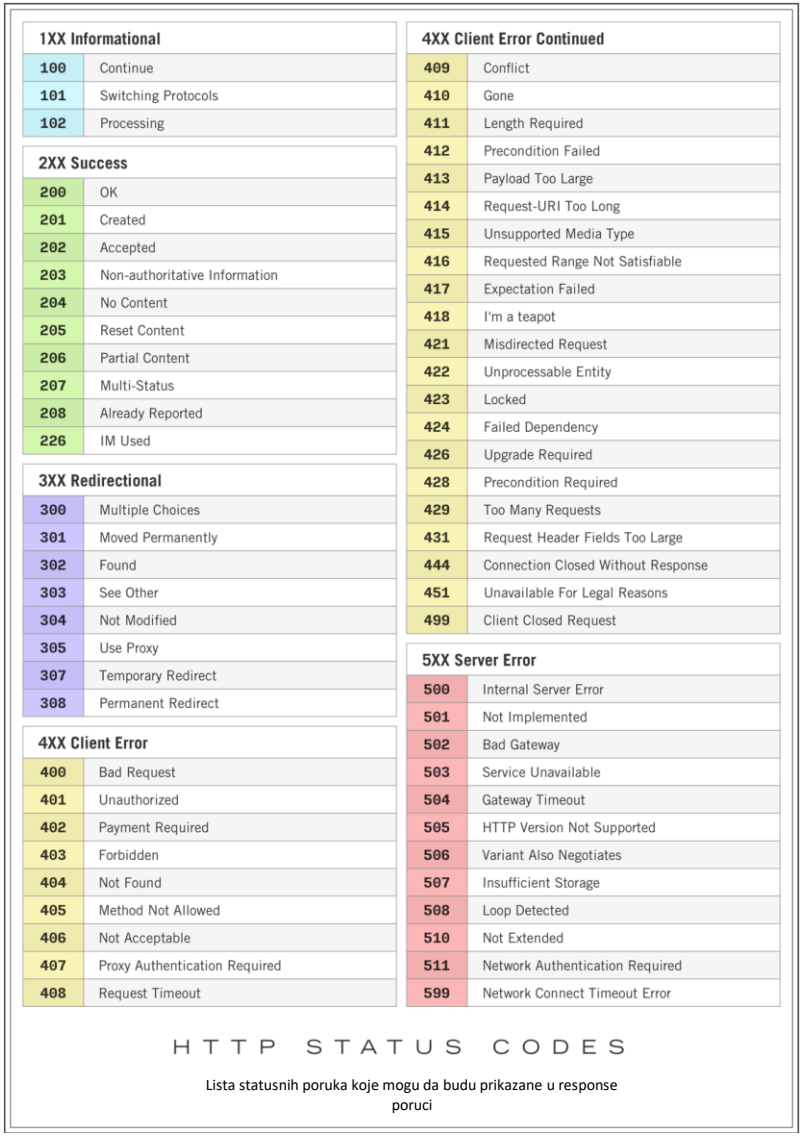
- Generička poruka o grešci, server nije razumeo request poruku.

404 Not Found

-Traženi dokument ne postoji na serveru.



STATUSNI KODOVI



1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

HTTP STATUS CODES

Lista statusnih poruka koje mogu da budu prikazane u response poruci

201 Created

- Uspešno je kreirana stranica ili update podataka

304 Not Modified

- Prikazan je keširani podatak koji je ranije posećen a nije promenjen.

401 Unauthorized

- Neautentifikovan pristup objektu (korisnik nije autentifikovan) u slučaju da nedostaje token za pristup objektu

403 Forbidden

- Autorizacija korisniku nije odobrena za pristup objektu tj. korisnik nema dozvolu za pristup objektu

409 Conflict

- Server ne može da obradi zahtev jer se javio konflikt prilikom pristupa resursu zbog višestrukog istovremenog editovanja.

500 Internal server error

505 HTTP Version Not Supported

- HTTP verzija nije podržana od strane servera.

HTTP RESPONSE PORUKA

```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Podaci u zaglavlju HTTP Response poruke

`Date:` – Vreme na serveru kada je kreirana respone poruka

`Server:` – Tip web servera koji je poslao poruku

`Last-Modified:` – Datum/vreme kada je objekat kreiran ili modifikovan

`Content-Length:` – Broj bajtova koji je poslat

`Connection:` – Server će zatvoriti TCP konekciju nakon što pošalje odgovor .

`Content-Type:` – Objekat u zaglavlju je HTML text

HTTP RESPONSE PORUKA

```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Telo poruke

- HTML text i drugi objekti koje razume browser na klijentu
- JSON (JavaScript Object Notation)
- XML (eXtensible Markup Language)
- Binarni fajlovi (slike I video snimci)
- Skripta

HTTP REQUEST \ RESPONSE PORUKA

HTTP
Server

HTTP REQUEST(1)

HTTP RESPONSE(2)

HTTP
Klijent

HTTP RESPONSE PORUKA

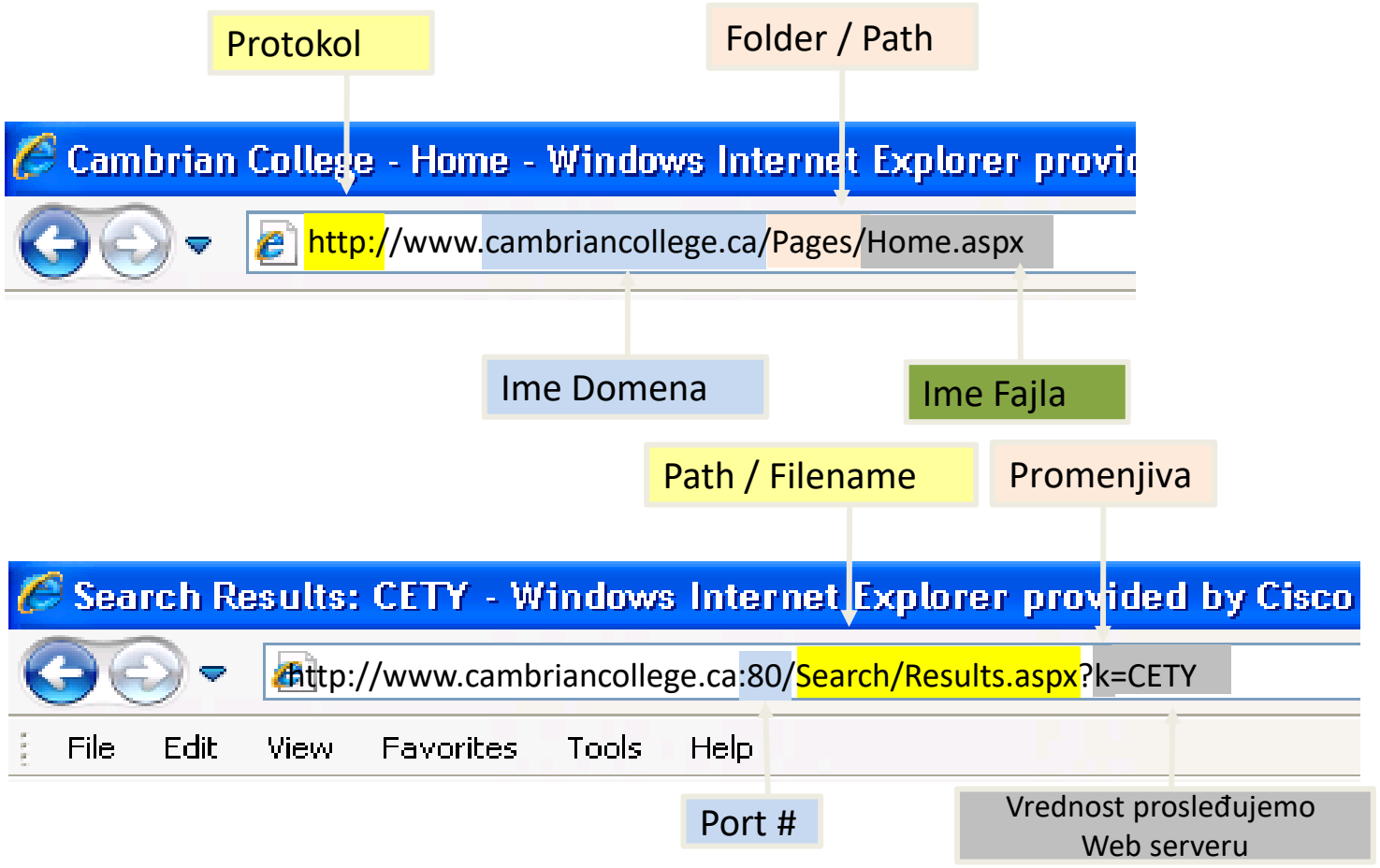
```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 16:34:18 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Thu, 15 Nov 2007 19:33:12 GMT
Content-Length: 15137
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

HTTP REQUEST PORUKA

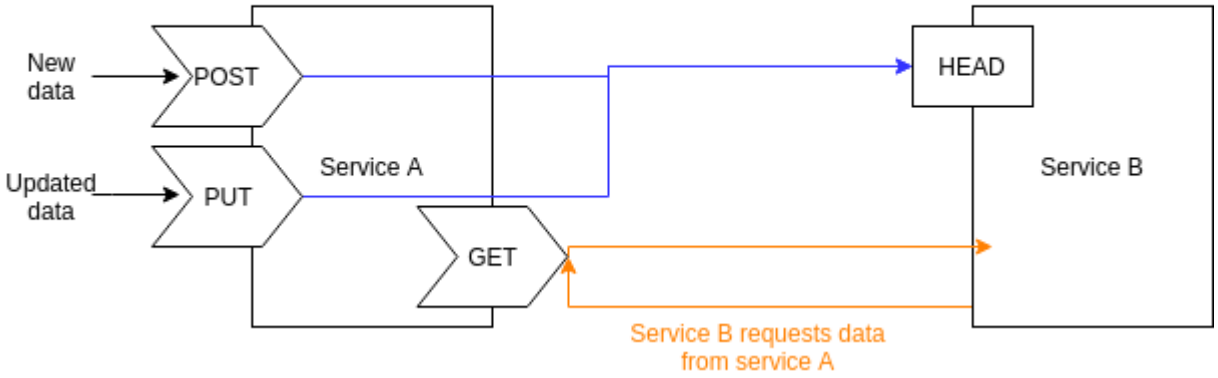
```
GET /~vtsnis/ HTTP/1.1
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE
    7.0; Windows NT 6.0; SLCC1; .NET CLR
    2.0.50727; Media Center PC 5.0; .NET
    CLR 3.0.04506; InfoPath.1)
Host: www.vtsnis.edu.rs
Connection: Keep-Alive
```

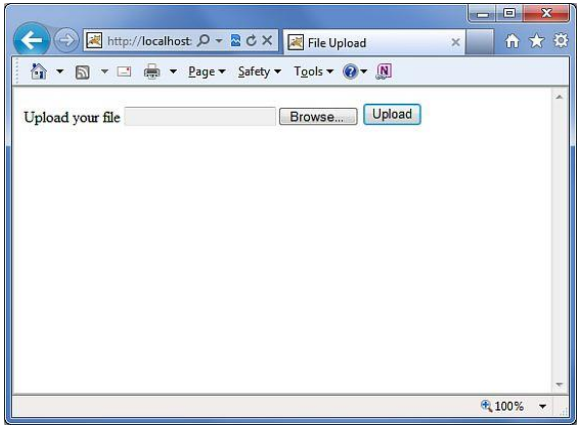
Struktura URL adrese

URL ili URI (Uniform Resource Identifier) je ime na osnovu koga se pristupa Web strani na Web serveru preko Web čitača



HTTP REQUEST KOMANDE





POST i **PUT** komande se koriste za slanje podataka (upload) na web server.

Kada korisnik unese podatak u formu koja je ugrađena u web strani, **POST** metodom šalje se podatak Web serveru.

PUT metodom se vrši ažuriranje podataka na web serveru, promena slike ili teksta na web sajtu.

DELETE metodom se brišu podaci na web server.

Autorizacione strategije na web-u

HTTP ne prati stanje sesije tj. novi reqest ne sadrži informacije iz prethodnog zahteva kao što je uspešno logovanje ili autorizacija za pristup resursu na web sajtu

Svaki zahtev se tretira kao nov zahtev koji nije povezan sa prethodnim, međutim dizajn HTTP zahteva je takav da oni zavise jedan od drugog (npr. dodavanjem stavki u korpu treba da se poveže sa našim nalogom).

Postoji više načina kako web aplikacija može da upravlja sesijama.

Dva najpopularnija su:

Session token - tradicionalna metoda

Server kreira log o sesiji (prati stanje sesije) i čuva zapis sve dok je sesija aktivna

Najčešći način za prosleđivanje informacije o sesiji je preko kolačića (cookies) koji se nalaze u zaglavlju HTTP poruke.

Informacija o sesiji se nalazi u cookie zaglavlju

Trenutno najpopularnija metoda za autorizaciju (session id + cookies)

Jason Web token (JWT) – modernija metoda

Web app kreira token koji čuva klijent i koji sadrži digitalni potpis web app koja je kreirala token.

Jason token koji šalje klijent nije samo ID već sadrži sve informacije o samom klijentu (Jason objekat)

Kreiranje Sesija

Najčešće upotrebljavanja metoda za praćenje sesija je korišćenje ID sesije koju kreira i podešava server.

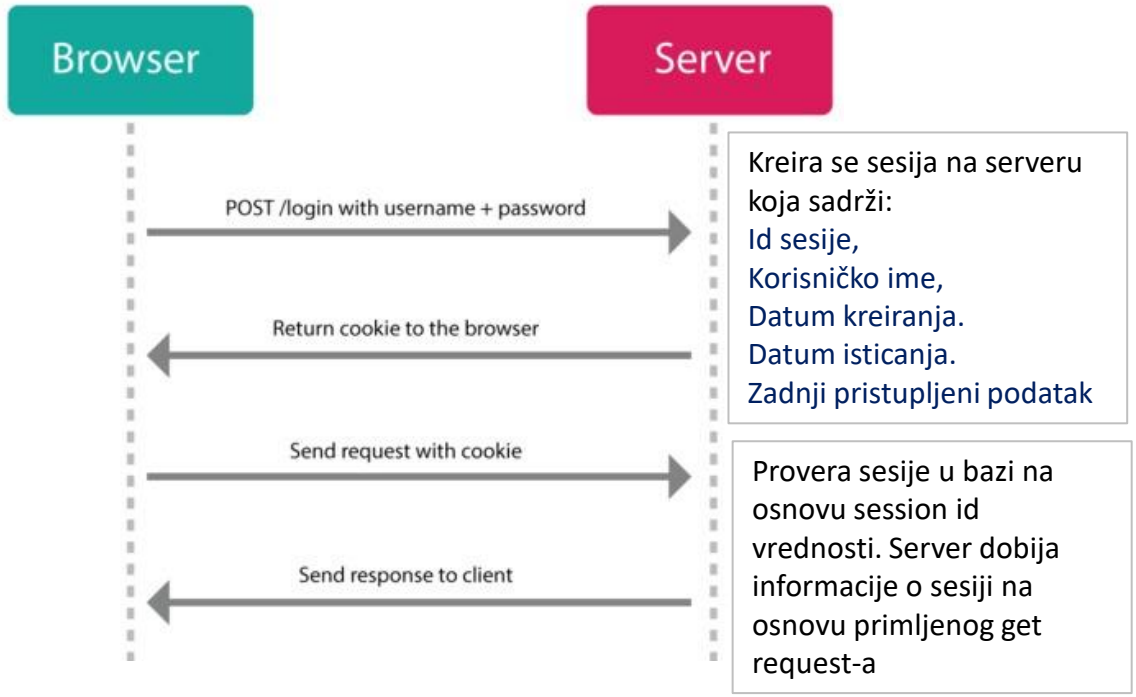
Čim se proveri identitet korisnika validnim korisničkim imenom i lozinkom, korisniku je dodeljen jedinstven nasumični ID sesije.

U svaki zahtev koji klijent pošalje uključen je ID sesije koji ima za cilj da poveže zahtev sa korisnikom čiji je identitet proveren.

ID sesije može da se šalje metodom GET ili POST. Ako se koristi metoda GET, ID sesije će postati deo URL adrese a kad se koristi metoda POST ID sesije se nalazi u telu poruke

Server održava tabelu za mapiranje korisničkih imena i dodeljene sesije.

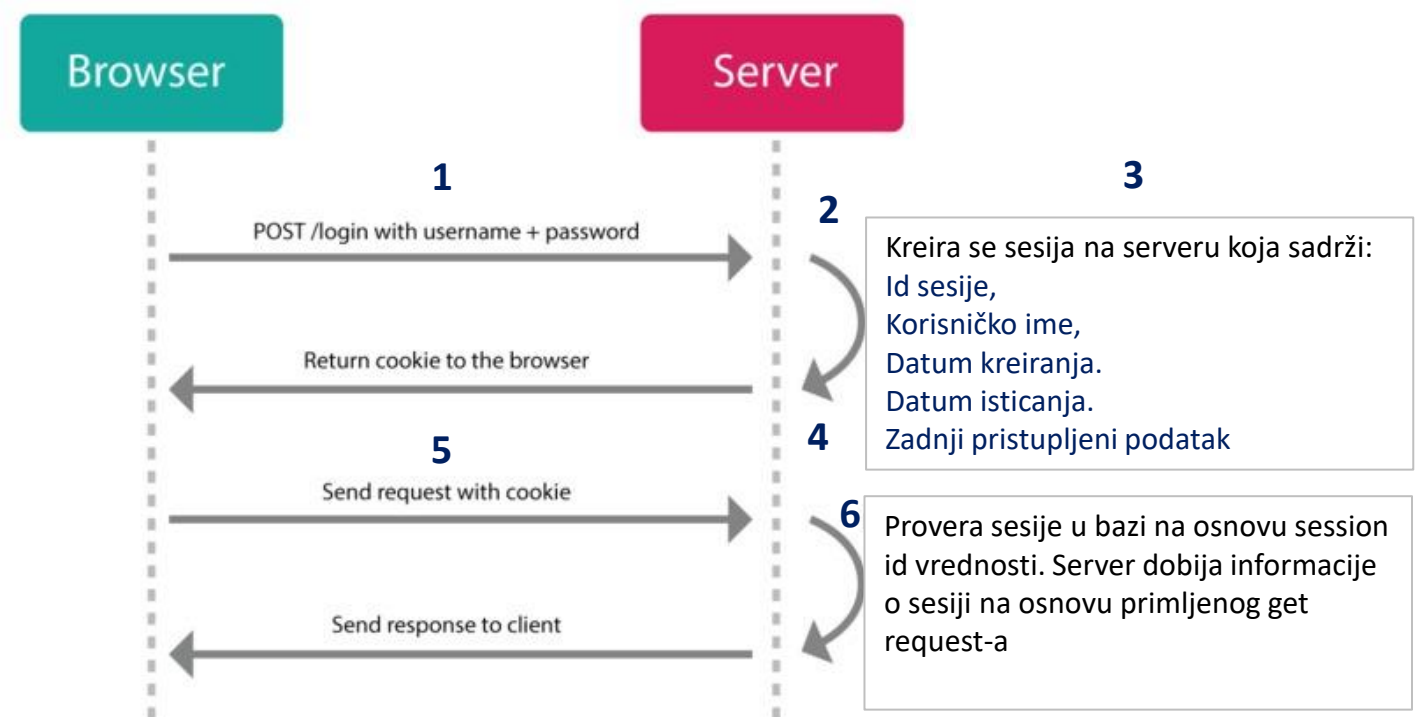
Na ovaj način get zahtevi koji dolaze od korisnika ne treba da se proveravaju svaki put.



Cookies – Praćenje autentifikovanog korisnika

Stateful auth (odnosi se na sesije)

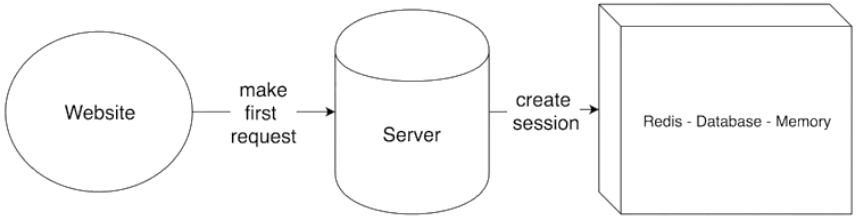
1. Korisnik šalje login kredencijale
2. Server proverava kredencijale u bazi
3. Server kreira privremenu tzv. korisničku sesiju
4. Server kreira **cookie za slanje informacije o kreiranoj sesiji** na osnovu Session ID
5. Korisnik šalje cookie koji se čuva na klijentskoj strani a sadrži Session id u svakoj request poruci
6. Server verifikuje na osnovu session id koji je prethodno kreirao i dozvoljava pristup
7. Kada se korisnik izloguje, server zatvara sesiju i briše kolačić.

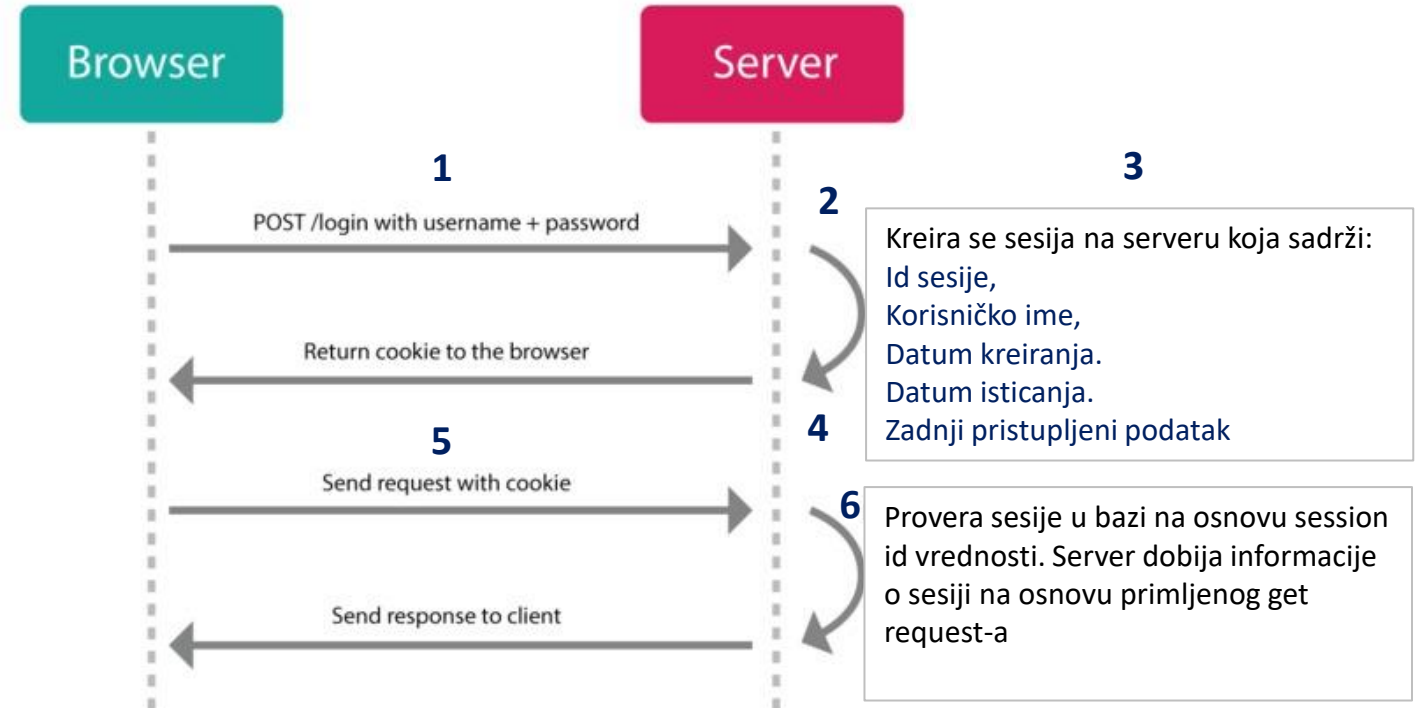


Cookies – Praćenje autentifikovanog korisnika

Svaka korisnička sesija se čuva **na serverskoj strani** (stateful)

1. Memoriji (fajl sistem) - retko
2. Keš (Redis ili Memcashed)-čest slučaj
3. Baza Podataka(Postgres ili MongoDb)



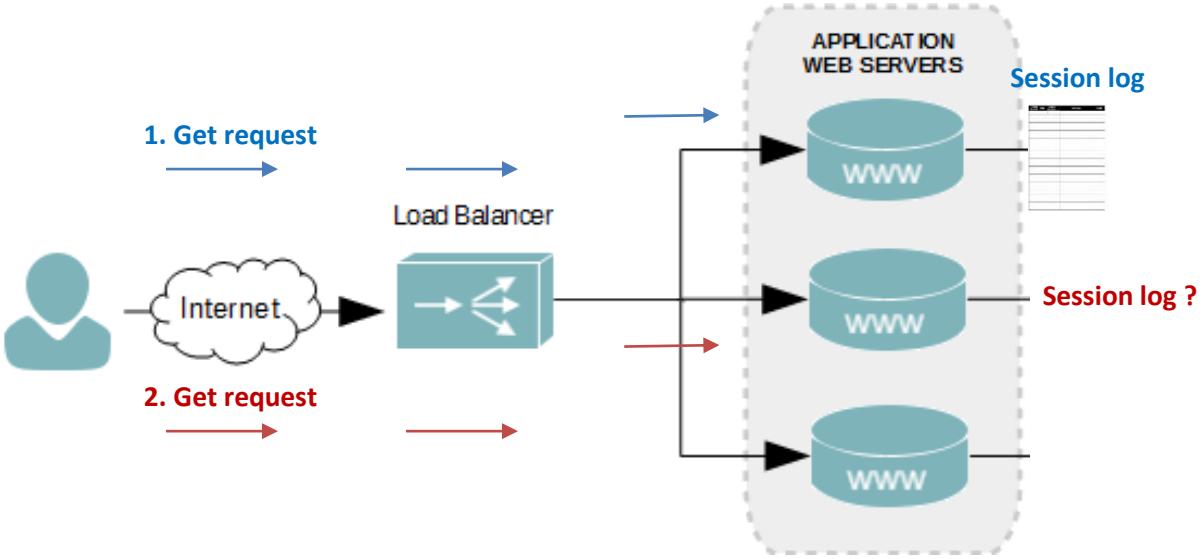


Session ID + Cookies – problem praćenja korisnika

Moderni web server zbog redudatnosti i bržeg opsluživanja klijenta sadrže više instanci tj. kopija.

Raspoređivači opterećenja (load balancer) kada primi zahtev od klijenta odlučuje kojoj www instanci će poslati zahtev

Problem se javlja kada se informacija o sesiji kreira na jednoj instanci a zatim neki od narednih zahteva load balanser prosledi drugoj instanci.



Session ID + Cookies – problem praćenja korisnika

Moguće rešenje

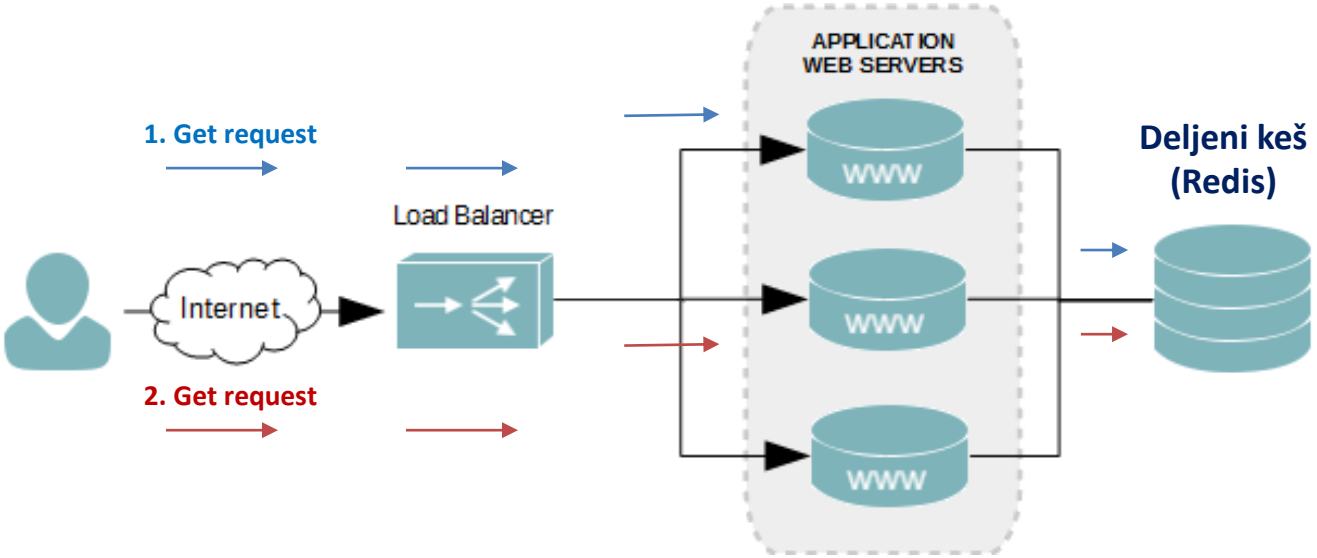
Kreira se in-memory baza podataka (Redis) koja čuva kreirane sesije svih instanci

Nedostatak je što je Redis *single point of failure* tj. ukoliko on otkaže sve informacije o sesijama se gube.

Alternativno rešenje je da se load balanser podesi da prati na kojoj instanci je kreirana koja sesija i da zahteve klijenata prosleđuje onoj instanci gde je kreirana njegova sesija.

Rešenje nije skalabilno

Drugi nedostatak je da onaj ko dobije pristup ID sesije može da se lažno predstavlja kao korisnik jer se od njega ne zahteva korisničko ime i lozinka.



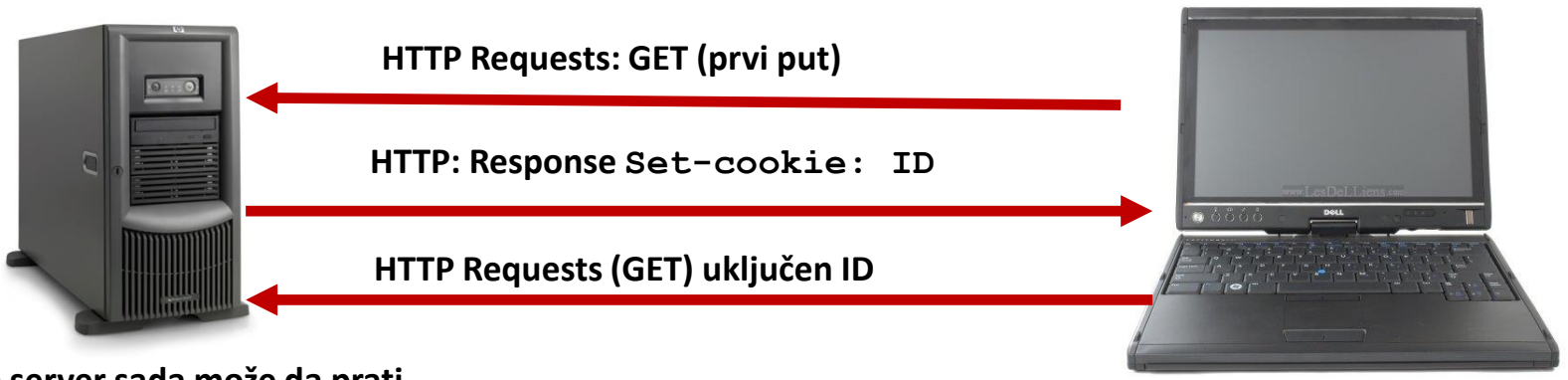
KLIJENT SERVER INTERAKCIJA PREKO KOLAČIĆA - COOKIES

Web serveri koriste kolačiće (**cookies)** za čuvanje informacija o sesiji.

Putem kolačića informacije o sesiji mogu da se prenosu do web servera.

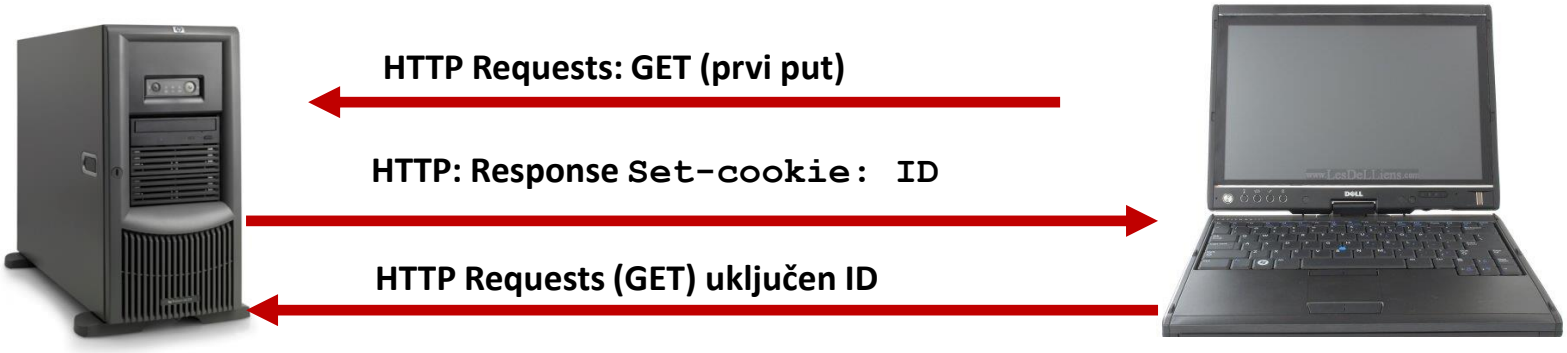
Najčešća metoda za prenos id sesije

Cookies su definisani RFC 2109



Web server sada može da prati aktivnost klijenta na web sajtu.

KOLAČIĆI (COOKIES)



Kolačić je deo informacije koji ima naziv, vrednost i parametre ponašanja

Kreira ga server a čuva se u fajl sistemu klijenta ili memoriji web pretraživača

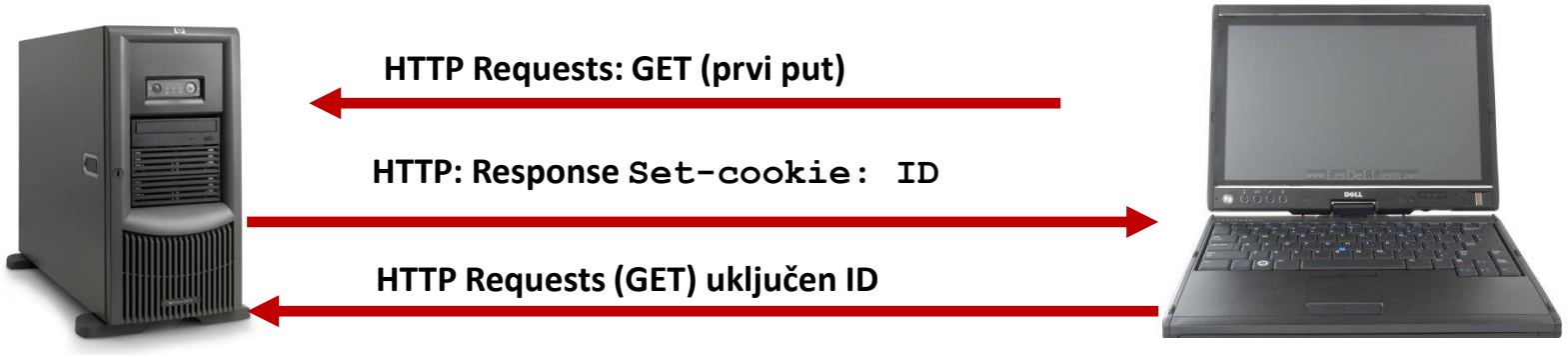
Vrednosti kolačića su povezane sa URL adresom.

Najčešći mehanizam prosleđivanja ID sesije između klijenta i servera je putem kolačića

Kada korisnik poseti originalni web sajt, pretraživač će poslati vrednosti kolaćića identifikujući korisnika.

Kolačići osim za praćenje sesija mogu da se upotrebe za čuvanje informacija za krajnjeg klijenta kao što su jezik i druge konfiguracione opcije.

KLIJENT SERVER INTERAKCIJA PREKO KOLAČIĆA - COOKIES



Web server instalira cookies na klijentu kada:

Pristupa web sajtu prvi put (Web server ne prepoznaje klijenta po imenu)

Korisnik saopšti informacije web serveru. (Web server prepoznaje klijenta po imenu)

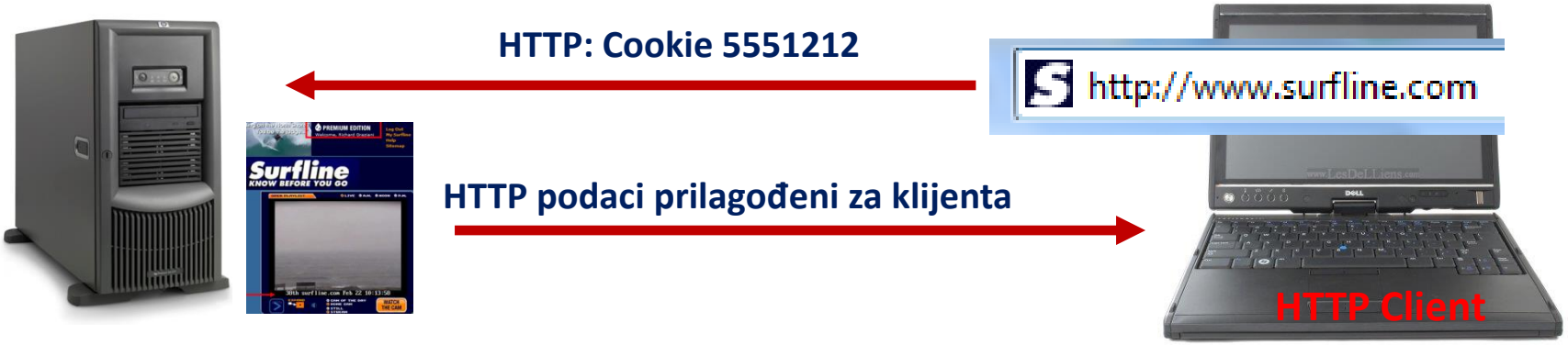
HTTP na Web serveru u zaglavlju odgovora uključuje **`Set-cookie:`** **ID**.

Ovaj **ID** se čuva na klijentskom računaru.

Svaki put kada client/browser pristupi web sajtu, **`GET`** metoda uključuje **`Cookie`** ili **`User_ID`**.

Kolačiće uvek kontroliše i postavlja server

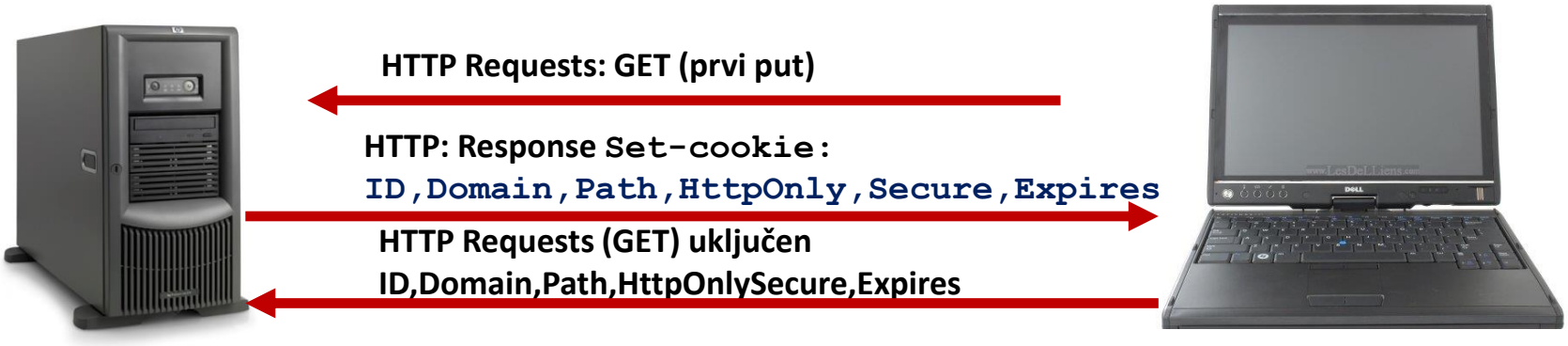
KLIJENT SERVER INTERAKCIJA PREKO KOLAČIĆA - COOKIES




```
HTTP/1.1 200 OK
Date: Fri, 22 Feb 2008 19:00:15 GMT
Server: Apache/1.3.34 (Unix)
Last-Modified: Fri, 22 Feb 2008
18:51:47 GMT
ETag: "760a31-18ce-47bf19c3"
Accept-Ranges: bytes
Content-Length: 6350
Keep-Alive: timeout=15, max=257
Connection: Keep-Alive
Content-Type: text/plain <......>
```

```
GET /jpeg/cap81/cam0.36705623.rgb888.enc HTTP/1.1
<.....................>
Cookie: SLSPOTNAME5=Cowells;
    SLSPOTNAME4=Waimea%20Bay;
    SLSPOTNAME3=Pipeline;
    SLSPOTNAME2=38th%20Ave%2E;
    SLSPOTNAME1=Cowells; SLSPOTID5=4189;
    SLSPOTID4=4755; SLSPOTID3=4750;
    SLSPOTID2=4191; SLSPOTID1=4189;
    OAX=R8bfwEbcU08ABCBu; USER_ID=5551212 <not my
    actual user-id>;........
```

Parametri kolačića



Web aplikacija može kolačiću da dodeli sledeće parametre:

Domain: sadrži domen kome se kolačić šalje

Path: putanja na serveru na kojoj će kolačić biti dostupan.

HttpOnly: sprečavamo da Java Script pristupi kolačiću a samim tim i potencijalni XSS (Cross site scripting) napad

Secure: zahteva slanje kolačića samo preko sigurnih prenosnih kanala (SSL/TLS)

Expires: definiše vreme brisanja kolačića

JSON WEB TOKEN

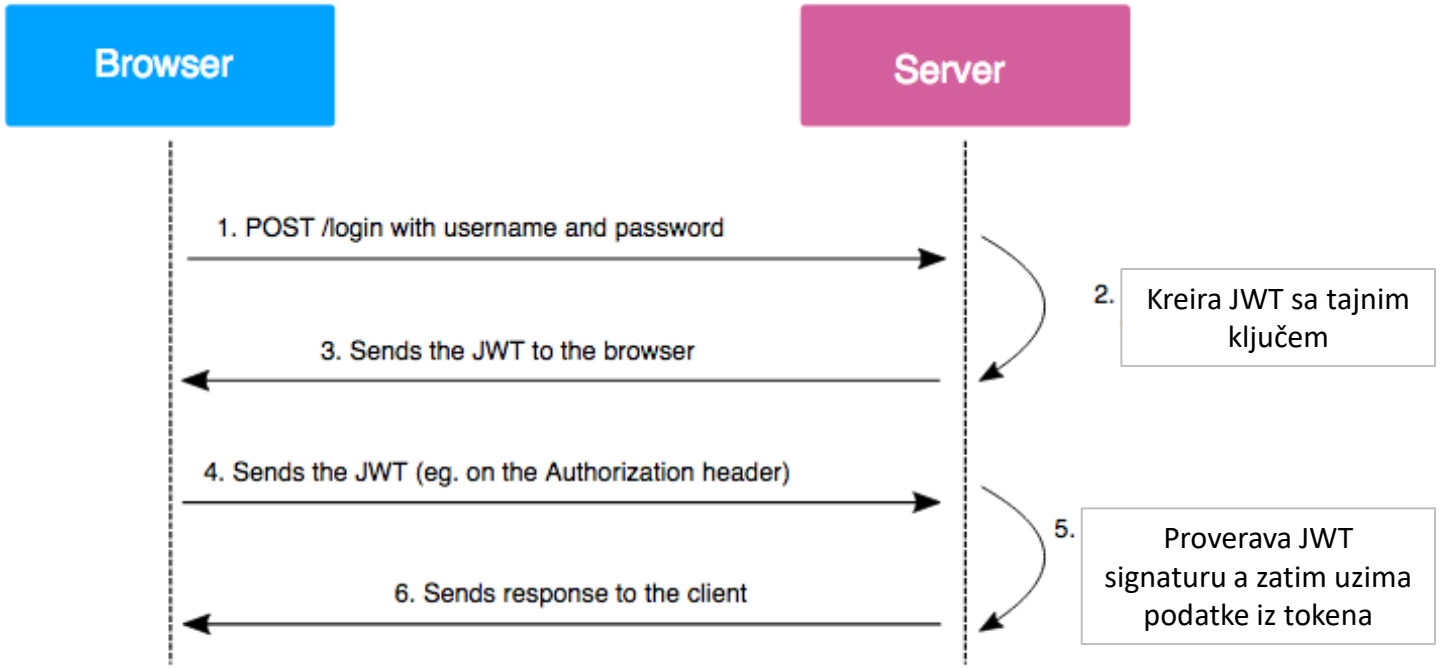
Koristi se za autorizacuju na web-u

Web aplikacija kreira token koji čuva klijent i koji sadrži digitalni potpis web aplikacije koja je kreirala token.

Jason token koji šalje klijent nije samo ID već sadrži sve informacije o samom klijentu (Jason objekat) jer je to način da server autentifikuje klijenta.

Problem bezbednosti je rešen jer je token potpisan.

JWT token se čuva na klijentu na disku (local storage ili u cookie) a šalje se kroz HTTP zaglavlje.



JSON WEB TOKEN

JWT token se najčešće sastoji iz tri dela koji su razdvojeni tačkom

Base64 enkoder se koristi da niz karaktera konvertuje u originalni niz

Payload su podaci u JSON formatu

Zaglavlje sadrži informaciju o algoritmu koj se koristi za kreiranje potpisa

Potpis služi da server verifikuje token tj. da su vrednosti tokena validne a ne da se token kriptuje. Samo server koji je kreirao token može proveri autentičnost tokena na osnovu kreirane signature i tajnog ključa koji samo on zna.

1 **Zaglavlje**
Algoritam i tip tokena

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 **Payload**
Podaci

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

3 **Signatura**
Digitalni potpis

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

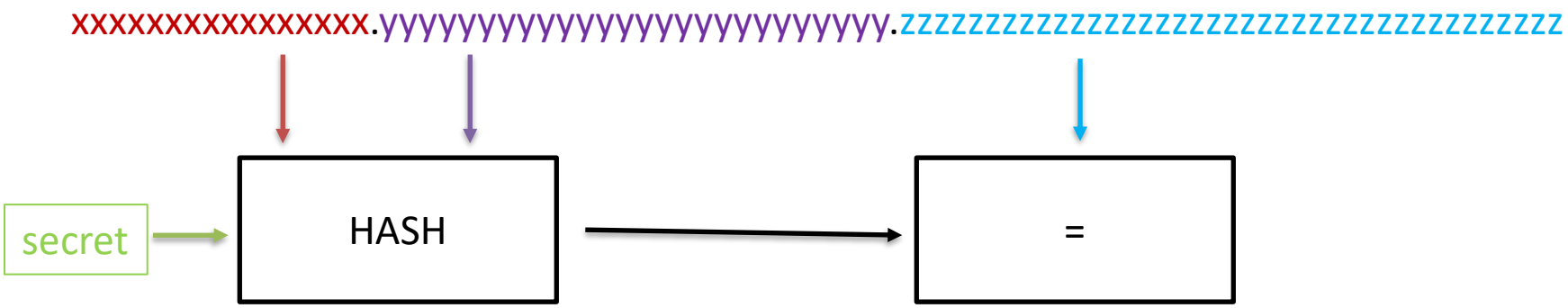
JSON WEB TOKEN – Verifikacija na serveru

JWT token ne mora da sadrži poverljive informacije kao što je lozinka već osnovne informacije da bi server znao o kome se radi npr. korisničko ime.

Ukoliko je JWT token ukraden, server neće znati da li je token stigao od prave osobe ili osobe koja je ukrala token. Zato se preporučuje da se vodi računa kako se token šalje preko mreže (HTTPS).

U odnosu na session token, ukoliko je došo do krađe session id-a čim se izlogujemo server briše sve informacije o sesiji.

JWT token se čuva na klijentu, server nema mogućnost da JWT token uništi tj. odbaci. Glavni nedostatak ukoliko dođe do zloupotrebe JWT tokena. Kao mogućnost postoji opcija da se na serveru čuva crna lista JWT tokena.



SLANJE JWT - HTTP REQUEST HEADER

TROSLOJNE WEB APLIKACIJE

Složenije Web aplikacije zahtevaju višeslojnu arhitekturu jer se dobijaju bolje performanse i veća bezbednost aplikacije.

U troslojnoj web aplikaciji postoji fizička odvojenost između slojeva:

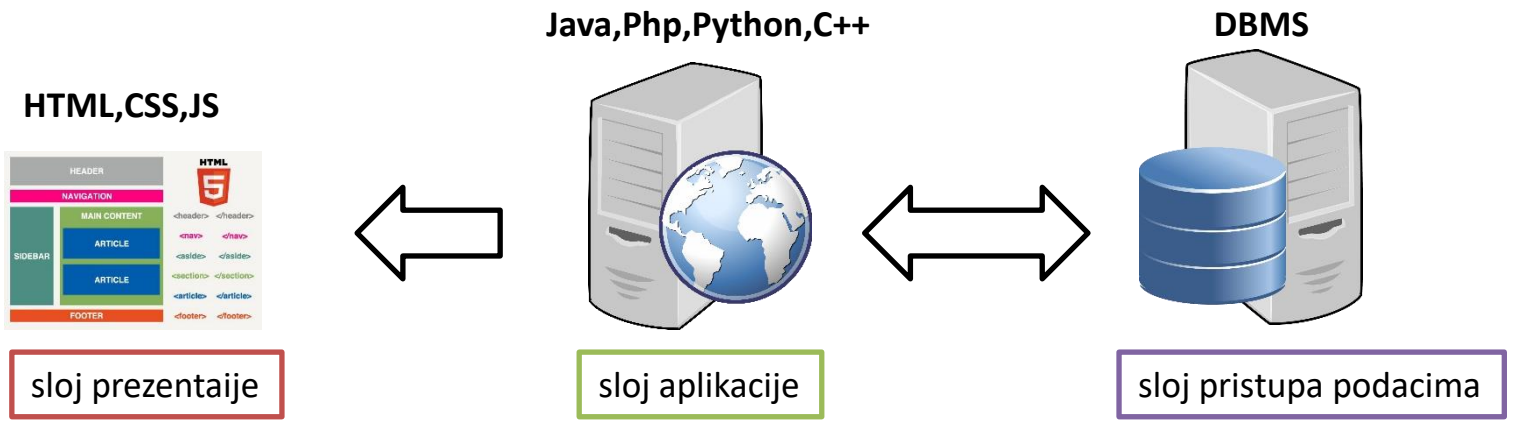
Sloj prezentacije:

Server koji prima zahteve klijenta i koji šalje odgovor klijentu.

Predstavlja interfejs između korisnika i ostatka aplikacije.

Podaci primljeni u sloju prezentacije se prosleđuju u komponente sloju aplikacije na obradu a primljeni rezultat je formatiran pomoću HTML-a i prikazuje se web klijentu korisnika.

Programi kao što su JS, React, Angular, VUE su spoređeni da rade u sloju prezentacije.



TROSLOJNE WEB APLIKACIJE

Sloj aplikacije:

Sadrži glavnu logiku aplikacije

Primena poslovne logike nad podacima primljenih sa sloja prezentacije.

Rezultat se vraća u sloj prezentacije a zatim odatle šalje klijentu.

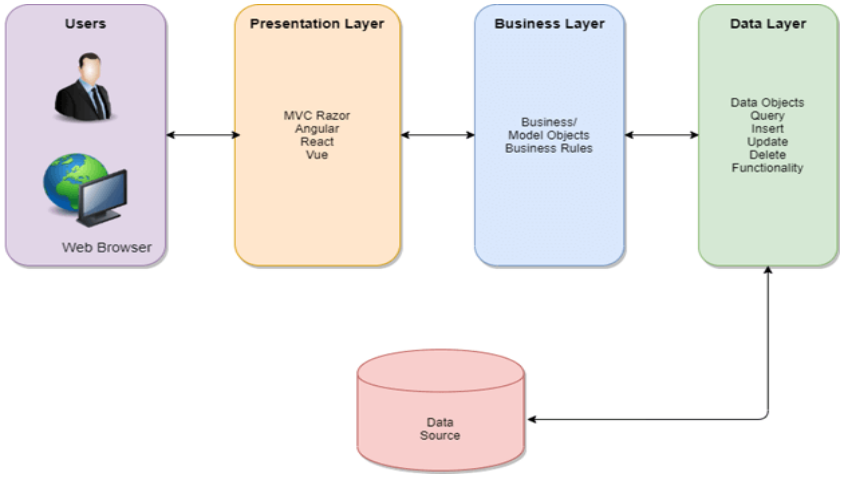
Programski jezici koji funkcionišu na ovom sloju su PHP, JAVA, Python i ASP.NET

Sloj pristupa podacima:

Komponente na ovom sloju su odgovorne za održavanje podataka, čuvanje integriteta i dostupnosti i upravljanje konkuretnim konekcijama iz sloja aplikacije.

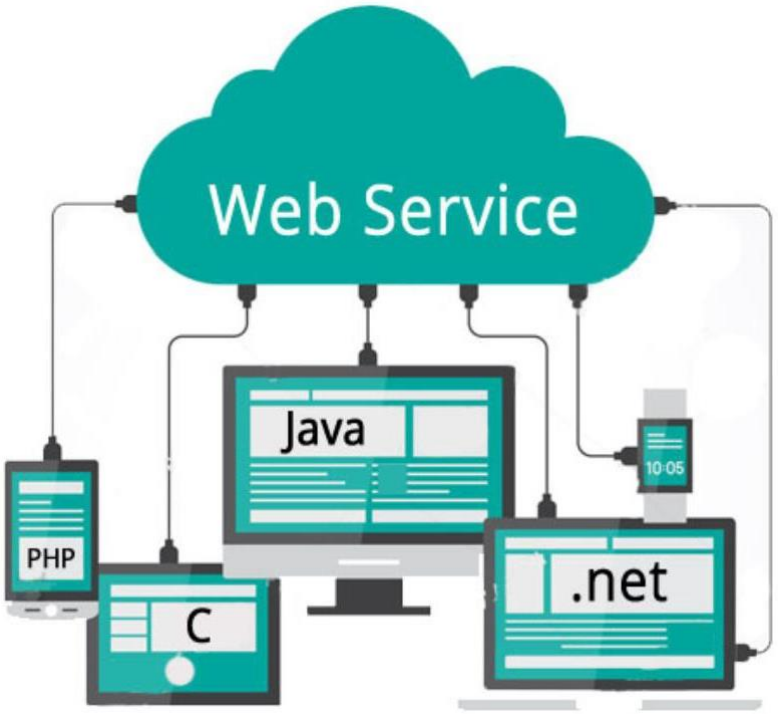
Relacione baze podataka (MySql i MS SQL) su dve najčešće tehnologije koje funkcionišu na ovom sloju.

Poslednjih godina javljaju se web aplikacije koje na sloju pristupa podacima koriste NoSql bazu podataka.



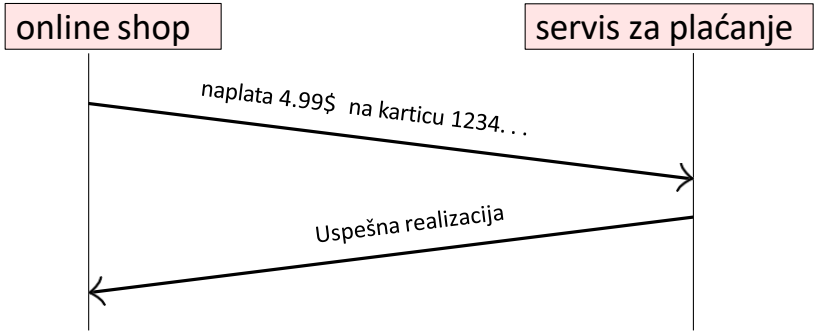
WEB SERVISI

- Web aplikacije koje ne uključuju sloj prezentacije
- Omogućavaju različitim aplikacijama da dele podatke i funkcionalnosti između sebe
- Arhitektura orjentisana ka servisu omogućava provajderu web servisa da se lako integriše sa drugim aplikacijama
- Servis ne otkriva logiku koja je upotrebljena za kreiranje podataka ali dozvoljava pristup podacima.
- Npr. Kladionice koje u realnom vremenu na svojim sajtovima prikazuju rezultate utakmica.
- Web servisi su **nezavisni od platforme** koja koristi uslugu servisa
- Slaganje između korisnika servisa i provajdera servisa mora da postoji oko **pravila za razmenu podataka**
- Postoje dva načina za razvoj web servisa
  - **Simple Object Access Protocol (SOAP)**
  - **Representational State Transfer (REST)**



Pozivanje udaljene procedure (RPC)

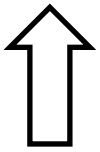
- Primer korišćenja web servisa je kupovina online proizvoda koji se plaća credit/debit karticom
- Web shop aplikacija koristi servis koji je specijalizovan za obradu plaćanja preko Interneta
  - Kada klijent pokrene kupovinu, parametri sa njegove kartice se šalju servisu koji je nezavisan od web shop aplikacije na obradu.
  - Servis za plaćanje (payment servis) ustvari komunicira sa infrastrukturom kartice (Visa, MasterCard,...) koja komunicira sa bankog gde je izdata kartica da bi se izvršilo plaćanje.



Primer koda za pozivanje udaljene procedure

- Kod za obradu plaćanja na strani online shop web aplikacije

```
// Online shop - upravljanje detaljima kartice
Card card=new Card();  card.setCardNumber("1234 5678 8765 4321");
card.setExpiryDate("10/2024");  card.setCVC("123");
Result result=paymentsService.processPayment(card,  4.99, Currency.dollar);
if(result.isSuccess())
{
  fulfilOrder();
}
```

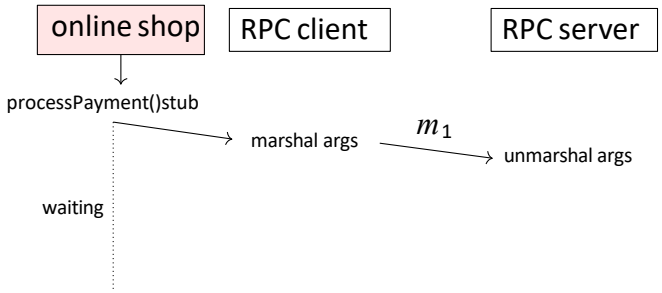


Implementacija Payment funkcije je na drugom čvoru

- Pozivanje funkcije za plaćanje iza scene šalje zahtev ka servisu za plaćanje, čeka odgovor i vraća odgovor
- Implementacija procesa samog plaćanja i sam kod se ne nalazi u web aplikaciji
  - To je deo servisa za plaćanje - drugi program koji se izvršava **na drugom čvoru** i pripada drugoj kompaniji.
- Vrsta interakcije gde kod na jednom čvoru poziva funkciju na drugom čvoru je Remote Procedure Call
- Software koji implementira RPC se zove **RPC framework ili middleware**
  - RPC framework **argumente funkcije enkodira** u poruku i šalje ih servisu za plaćanje

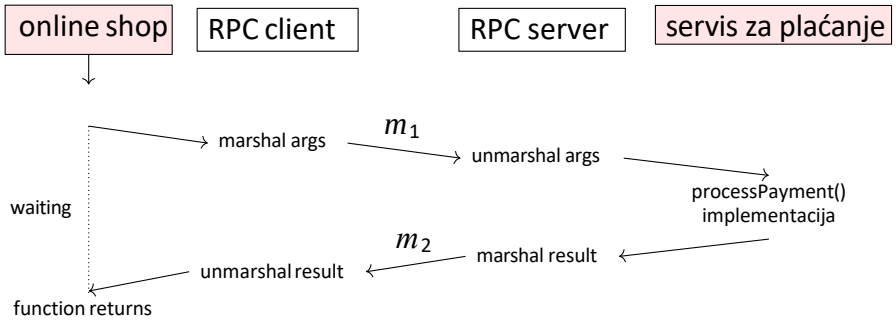
Pozivanje udaljene procedure (RPC)

- Proces enkodiranja argumenata funkcije se zove ***marshalling*** *i najčešće korišćeni format je JSON*
- *Slanje poruke sa RPC klijenta na RPC server obično je preko HTTP protokola (web servis)*
- *Na serverskoj strani RPC framework dekodira poruku i poziva željenu funkciju sa poslatim argumentima*
- *Kada funkcija vrati rezultat, on se enkoduje šalje u dogovorenom formatu klijentu*
- *Pozivaocu funkcije izgleda da se funkcija izršila lokalno*



m_1 =

```
{
  "request":"processPayment",
   "card": {
    "number":"1234567887654321",
    "expiryDate":"10/2024",
    "CVC":"123"
  },
  "amount":4.99,
  "currency":"DOLLAR"
}
```

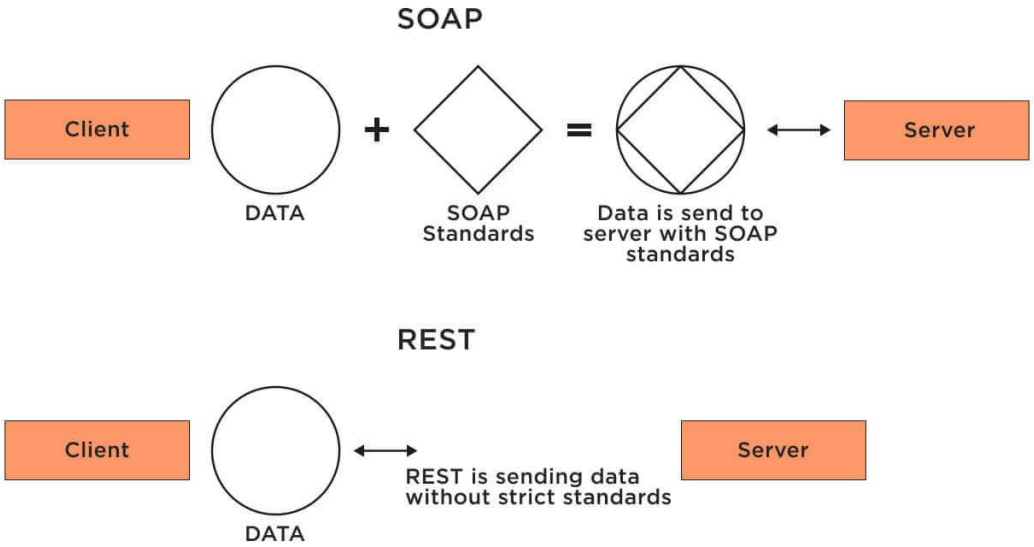


m_2 =

```
{
  "result":"success",
  "id":"XP61hHw2Rvo"
}
```

SOAP Vs REST web servis

- SOAP je tradicionalni metod za razvoj web servisa
  - XML format je jedini podržan format za razmenu podataka kada se koristi SIAP servis
  - SOAP je protokol i nezavisan je za transport
- REST web servis jednostavniji za implementaciju, direktno komunicira sa provajderom servisa pomoću HTTP-a bez potrebe za dodatnim protokolom.
  - REST web servisi mogu da koriste JSON i druge formate podataka
  - REST je zasnovan na HTTP-u i koristi HTTP za transport
  - REST servis koristi HTTP za čitanje,kreiranje, ažuriranje i brisanje podataka.



REST API

- RPC se danas najčešće implementira koristeći slanje podataka u JSON formatu preko HTTP protokola
- Popularni skup principa u dizajnu za takve API-je zasnovan na HTTP-u poznat je kao REST(*representational state transfer*) a API-ji koji se pridržavaju ovih principa nazivaju se RESTful.
  - Komunikacija je bezkonekciona (stateless), svaki zahtev je samostalan i nezavisan od ostalih
  - Resursi (objekti nad kojima se manipuliše) se predstavljaju URL adresama
  - Stanje objekta se ažurira preko HTTP zahteva standardnom metodom POST ili PUT do odgovarajuće URL adrese
- Popularnost REST API leži u korišćenju JS koda koji se danas izvršava u svim Web čitačima a koji olakšava kreiranje HTTP zahteva ka serveru.

REST API

- Kod uzima argumente, prevodi u JSON format koristeći funkciju JSON.stringify() i šalje na URL adresu https://example.com/payments preko HTTP POST zahteva

```
Let args={amount:4.99, currency:'DOLLAR',                    /*...*/};
let request={
    method:'POST',
    body:JSON.stringify(args),
    headers:{'Content-Type':'application/json'}
};

fetch('https://example.com/payments', request)
    .then((response) => {
        if(response.ok) success(response.json());
        Else failure(response.status);                  // server error
    })
    .catch((error) => {
        failure(error); // network error
    });
```

- Dva moguća scenarija
  - Server može da vrati status da je zahtev uspešno obrađen (response.json() metode čita primljeni json fajl) pozivajući success funkciju
  - Server može da vrati status da je zahtev nije uspešno obrađen ili da zahtev ne može da se isporuči do servera
- RESTful API i RPC zasnovan na HTTP su na webu i koriste se često ne samo u slučaju kada je JS klijent već u server server komunikaciji i mobilnim aplikacijama

RPC u distribuiranim sistemima

Server to Server RPC se primenjuje u distribuiranim sistemima gde je softver isuviše složen i veliki da se izvršava na jednoj mašini

"Service-oriented architecture" (SOA) / "microservisi":

- Razdvajanje složenog softvera u više servisa
- Na više čvorova koji komuniciraju preko RPC-a.
- Različiti timovi mogu da održavaju takav softver i da bude napisan na različitim programskim jezicima

Različiti servisi su pisani u različitim tehnologijama

- **Interoperabilnost** – konverzija tipova podataka da se poslati argumenti kompatibilni sa funkcijom koja se poziva na drugom čvoru i da su vraćeni podai mogu da se pročitaju
- **Interfejs za desinisanje jezi**ka (IDL) – jezik nezavisan od API specifikacije

Primer Interfejsa za definisanje jezika (IDL) gRPC

```
messagePaymentRequest{  messageCard{
      required string cardNumber=1;
      optional int32 expiryMonth=2;
      optional int32 expiryYear=3;
      optional int32 CVC=4;
  }
  Enum Currency {EUR=1;USD=2; }

   required Card card=1;
   required int64 amount=2;
   required Currenc ycurrency=3;
}

messagePaymentStatus{
Required bool success=1;
Optional string errorMessage=2;
}

Service PaymentService {  rpc ProcessPayment(PaymentRequest) returns (PaymentStatus) {}
}
```

XML i JSON formati podataka

- Formate XML i JSON koriste Web servisi za predstavljanje struktuiranih skupova podataka i objekata

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

XML Vs JSON

Čitljivost

- JSON – jednostavniji za čitanje jer se zasniva na objektima
- XML – složeniji za čitanje jer su podaci sadržani u tagovima

Kod

- JSON – sadrži manje koda
- XML – sadrži više koda

Brzina parsiranja

- JSON – Brži od XML-a, gde su podaci jasno definisani u obliku ključ i vrednost
- XML – Sporiji je jer podatke treba izvući iz tagova

Jednostavnost kreiranja

- JSON – Jednostavnija implementacija jer je sintaksa jednostavnija
- XML – Neznatno složeniji kod

Fleksibilnost i proširivost

- JSON – Podržava limitirani opseg tipova podataka, može biti nedovoljno za određene app
- XML – Slično programiranju

Bezbednost

- JSON – podskup JavaScript-a i kao takav može da bude iskorišćen za pokretanje malicioznog koda
- XML – je bezbedniji od JSON

JSON

```
{  "empinfo" :
    {
        "employees" :  [
        {
            "name" : "James Kirk",
            "age" : 40,
        },
        {
            "name" : "Jean-Luc Picard"
            "age" : 45,
        },
        {
            "name" : "Wesley Crusher
            "age" : 27,
        }
                    ]
    }
}
```

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

AJAX (Asynchronous JavaScript and XML)

- Kombinuje više web tehnologija, koja omogućava klijentu da pošalje zahteve i obrađuje odgovore u pozadini bez direktne intervencije korisnika.
- Omogućava komunikaciju sa web serverom bez potrebe da korisnik kreira eksplicitno novi zahtev u web pretraživaču
- Ažurira se web stranica bez učitavanja stranice
- Šalju se podaci na server u pozadini
- Rezultat je brži odgovor sa servera jer delovi web stranice mogu da budu ažurirani posebno
- AJAX koristi JS za povezivanje i preuzimanje informacija sa servera bez ponovnog učitavanja kompletne web stranice
- AJAX kombinuje XMLHttpRequest objekat koji je ugrađen u browser kojim šalje zahtev web serveru i JavaScript i HTML DOM za prikaz i korišćenje podataka
- AJAX aplikacije mogu da koriste XML za transport podataka, češće se za transport podataka koristi čist tekst ili JSNON format.

Osobine AJAX tehnologije

1. *Povećanje brzine – ažuriranje pojedinačnih elemenata zahteva minimalnu obradu*

2. Jednostavnost – aplikacija zasnovana na AJAX-u ne zahteva ponovno učitavanje cele strane da bi se odradio refresh specifičnih delova web sajta to aplikaciju čini interaktivnijom jednostavnijom

3. Asihroni pozivi – aplikacije zasnovana na AJAX-u su dizajnirane da kreiraju asihrone pozive za web server, na taj način se obavlja interakcija sa korisnikom dok se jedan njen deo ažurira. Web stranica se ažurira asihrono razmenom podataka sa web serverom u pozadini. Na taj načim moguće je ažurirati delove web strane bez učitavanja cele strane.

4. Smanjena upotreba mreže – Neizvršavanjem potpunog osvežavanja stranice smanjena je i upotreba mreže. U web aplikacijama u kojima postoje velike slike, video materijal ili dinamički sadržaj upotreba AJAX-a može da optimizuje upotrebu mreže.

5. AJAX dobro poznati radni okviri (fremork) - Jquery, Dojo Toolkit, Google Web Toolkit (GWT) i Microsoft AJAX biblioteka

Komponenta AJAX-a

JavaScript

Vrši interakciju sa web serverom u pozadini i obrađuje informacije pre nego što su one prikazane korisniku

JS koristi XMLHttpRequest (XHR) API za prenos podataka između servera i klijenta koji se izvršava u pozadini.

Dinamički HTML (DHTML)

Kada su podaci preuzeti sa servera i JS ih obradi, elementi web stranice treba da budu ažurirani da bi reflektovali odgovor sa servera.

Primenom DHTML-a i JS može se ažurirati sadržaj stranice u toku rada.

Osnovni nedostatak upotrebe samo DHTML-a je što on zavisi od koda na stranici klijenta za ažuriranje stranice. Uglavnom se vrši interakcija sa kodom na strani servera. AJAX je taj koji kreira konekciju između koda na strani klijenta i koda na strani servera.

Document Object Model (DOM)

DOM je upotrebljen za organizovanje elemenata u HTML ili XML dokumentu.

AJAX komunikacija između Web browser-a i Web servera

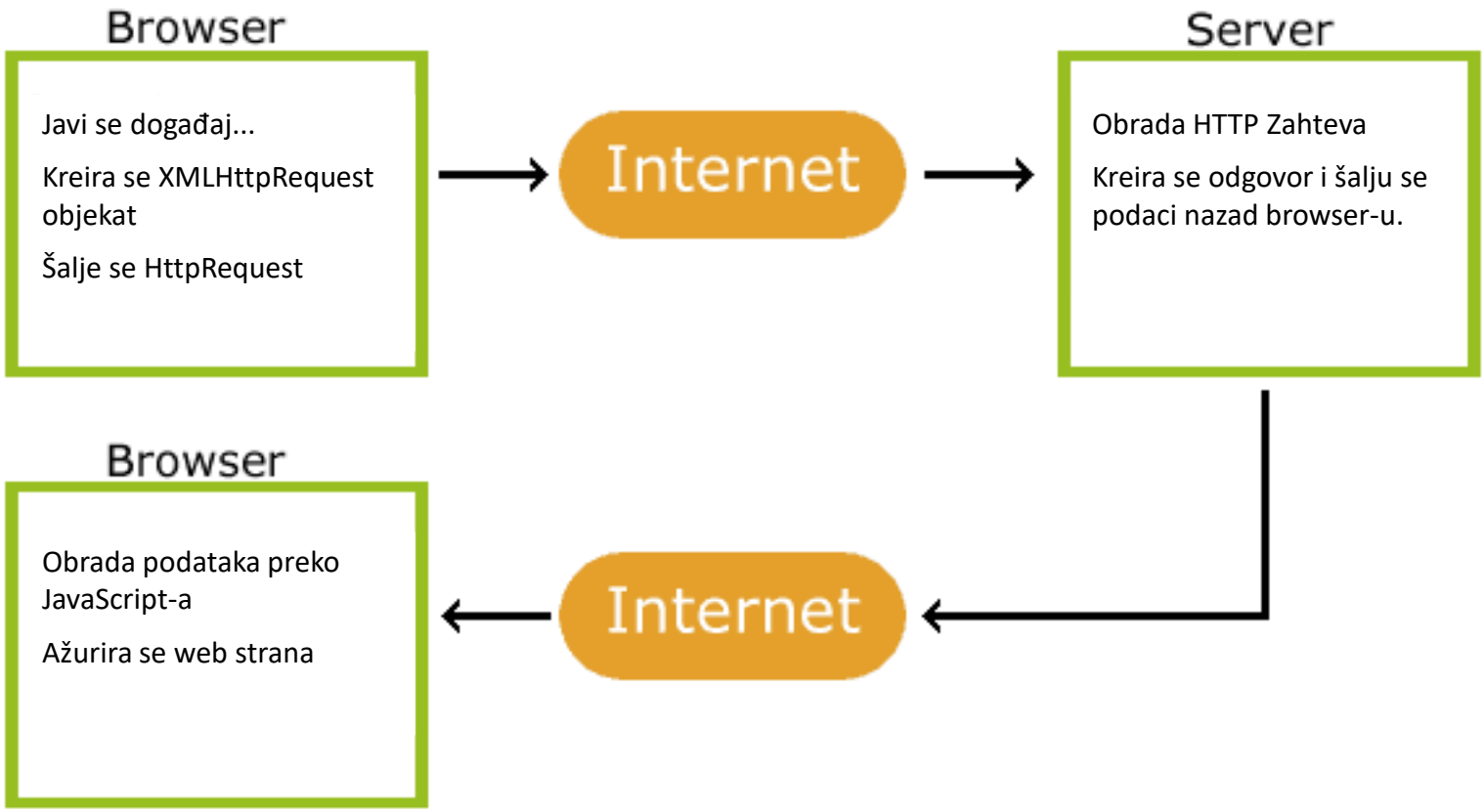
Korak 1

Korisnik unosi URL web stranice

Pretraživač šalje HTTP zahtev web serveru

Server obrađuje zahtev i vraća nazad odgovor sa HTML sadržajem koji je prikazan u pretraživaču.

Web stranica je ugrađena u JavaScript kod koji izvršava JS interpreter kada se desi događaj



Tok rada AJAX-a

Korak 2

Kada korisnik vrši interakciju sa web stranicom, korisnik se susreće sa elementom koji koristi ugrađeni JavaScript kod i pokreće događaj.

Npr. Google pretraga, čim korisnik krene da unose upit za pretragu u pozadini AJAX engine presreće zahtev korisnika i prosleđuje zahtev na server pomoću HTTP zahteva

Zahtev je transparentan korisniku, korisnik ne treba da klikne na dugme za generisanje zahteva ili da osvežava stranicu.

Korak 3

Web server obrađuje zahtev i vraća podatke nazad u AJAX engine u JSON, HTML ili XML formatu.

Ajax engine na klijentu prosleđuje podatke u engine za renderovanje web-a i prikazuuju se u web pretraživaču koji koristi DHTML za ažuriranje samo označenog segmenta