



SVEUČILIŠTE U ZAGREBU



Fakultet  
elektrotehnike i  
računarstva

**Diplomski studij**

**Informacijska i  
komunikacijska tehnologija:**

Telekomunikacije i informatika

**Računarstvo:**

Programsko inženjerstvo i  
informacijski sustavi

Računarska znanost

# Raspodijeljeni sustavi

## 3. Arhitekture web-aplikacija, tehnologije weba

Ak. god. 2020./2021.

# Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

# Sadržaj predavanja

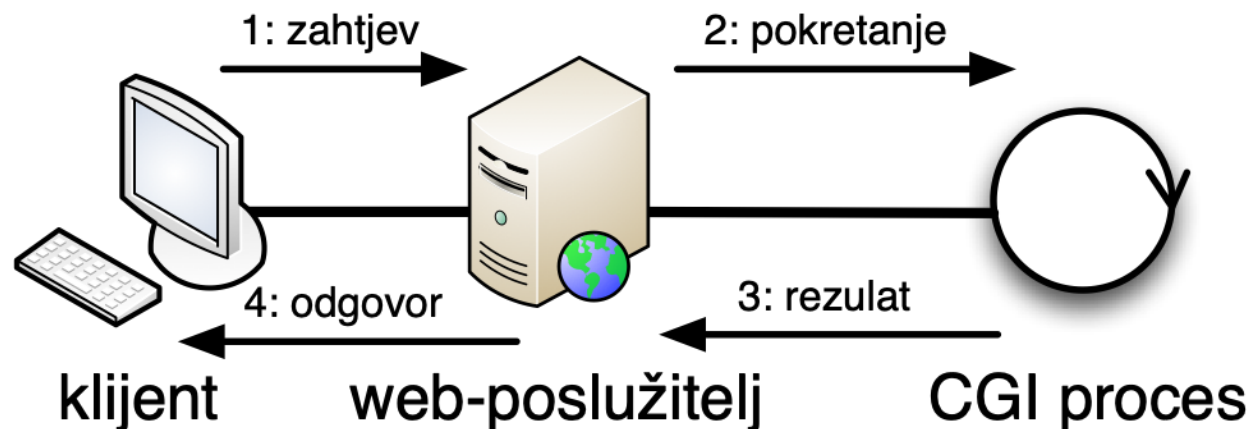
- arhitekture web-aplikacija
- AJAX
- uvod u Web 2.0
- Web-usluge
- jezici i protokoli: SOAP, WSDL, UDDI
- web-usluge temeljene na RPC-u, temeljene na dokumentima
- Web-usluge temeljene prijenosu prikaza stanja resursa (REST)
- Svojstva metoda protokola HTTP
- Model zrelosti web-usluga i relevantni standardi
- Implementacija REST-a u Springu

# Web-aplikacije

- Definicija:
  - *"Web applications are stored on web servers, and use tools like databases, JavaScript (or Ajax or Silverlight), and PHP (or ASP.Net) to deliver experiences beyond the standard web page or web form."*
- dvije vrste:
  - koje izgledaju kao normalne web-stranice (npr. portali)
  - koje izgledaju kako normalne aplikacije - bogato korisničko sučelje (npr. Google Mail)
- koriste tehnologije weba:
  - HTML, CSS, JavaScript, PHP, ASP, JSP, Ruby on Rails, Java Servlets, Cold Fusion, ...

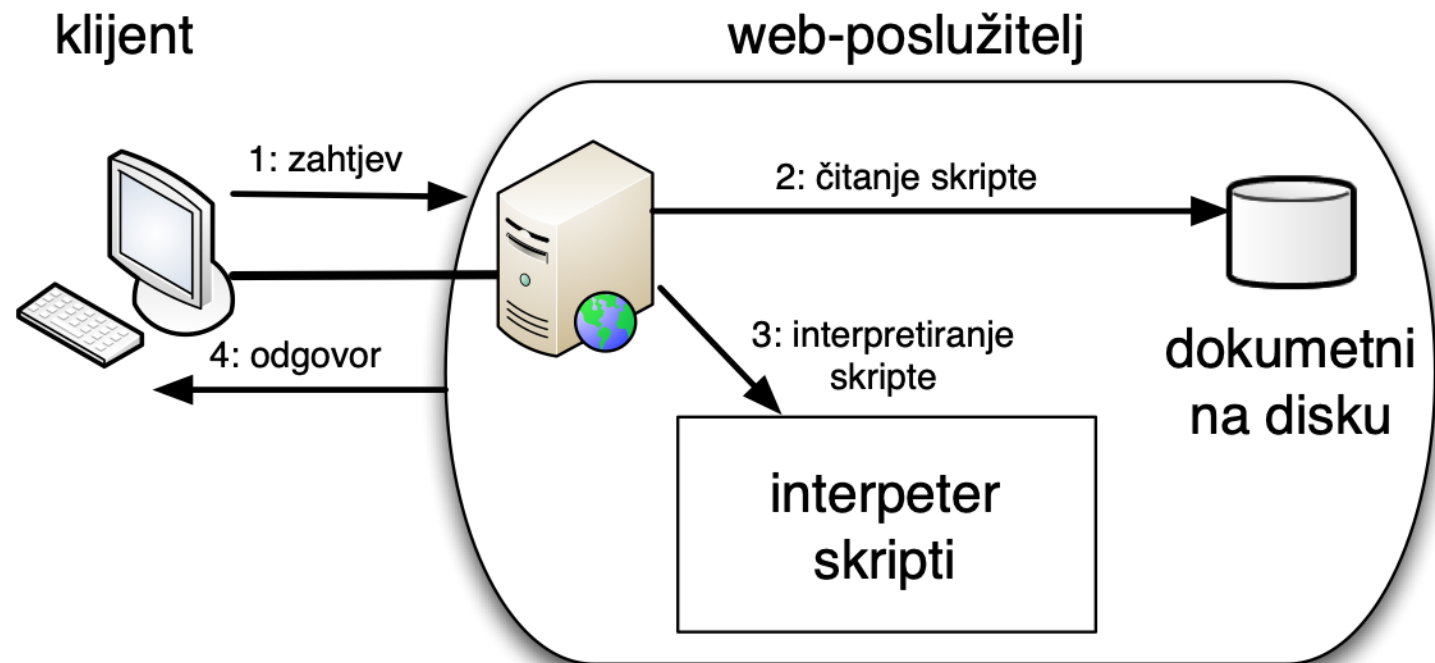
# Web-aplikacije - CGI

- CGI - *Common Gateway Interface* - RFC 3875
- kod svakog zahtjeva se pokreće proces
- podaci između poslužitelja i procesa se šalju preko varijabli okoline i tokova podataka
- nakon svake obrade proces se gasi
- Bash, Perl i ostali skriptni jezici



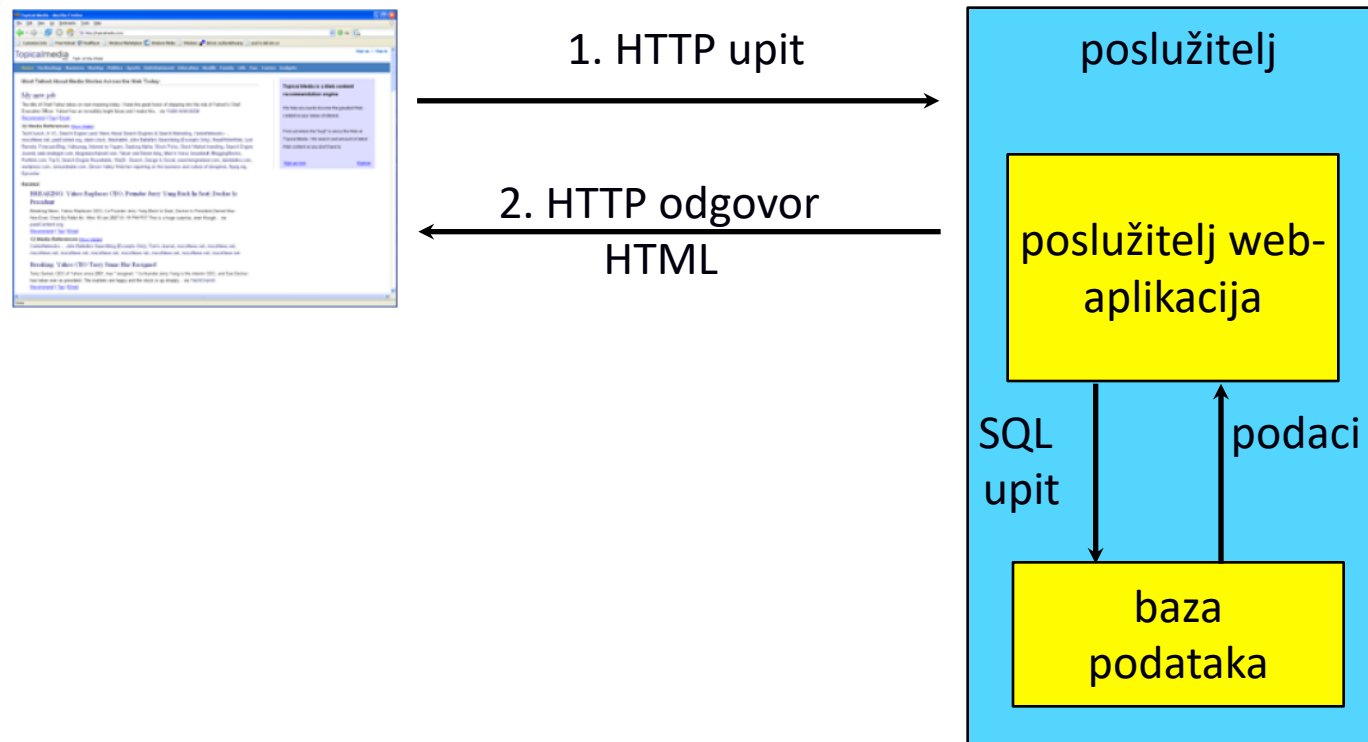
# Web-aplikacije - poslužiteljske skripte

- poslužiteljske skripte - *server side scripts*
- dinamički se generira HTML-dokument (iz skripte)
- primjeri: PHP, ASP, JSP, Django, Ruby on Rails, ...



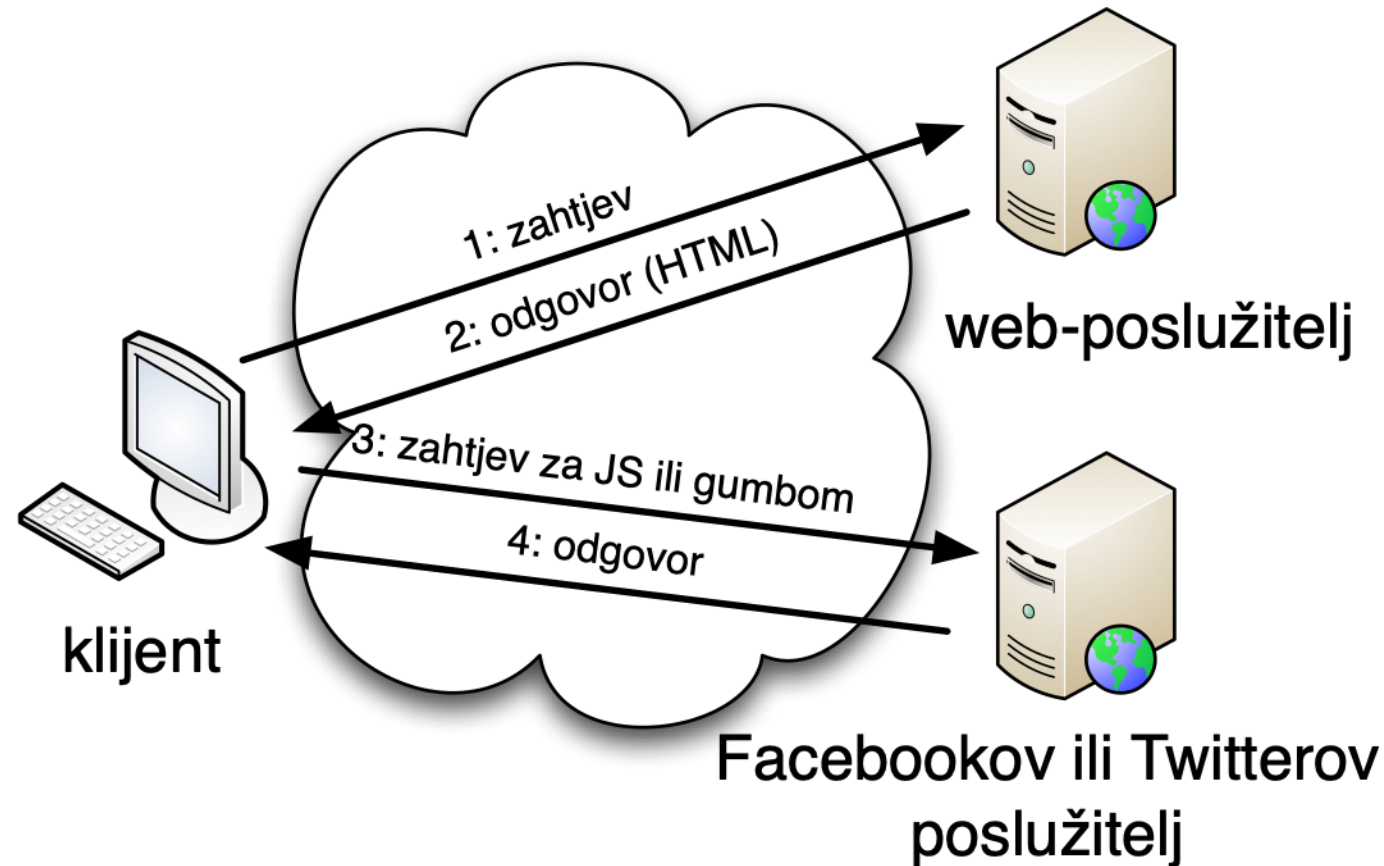
# Tipična web-aplikacija na jednom računalu

- razvijateljska konfiguracija
- dobro za mali broj zahtjeva u produkciji



# Web-aplikacija integrirana s drugim sjedištima

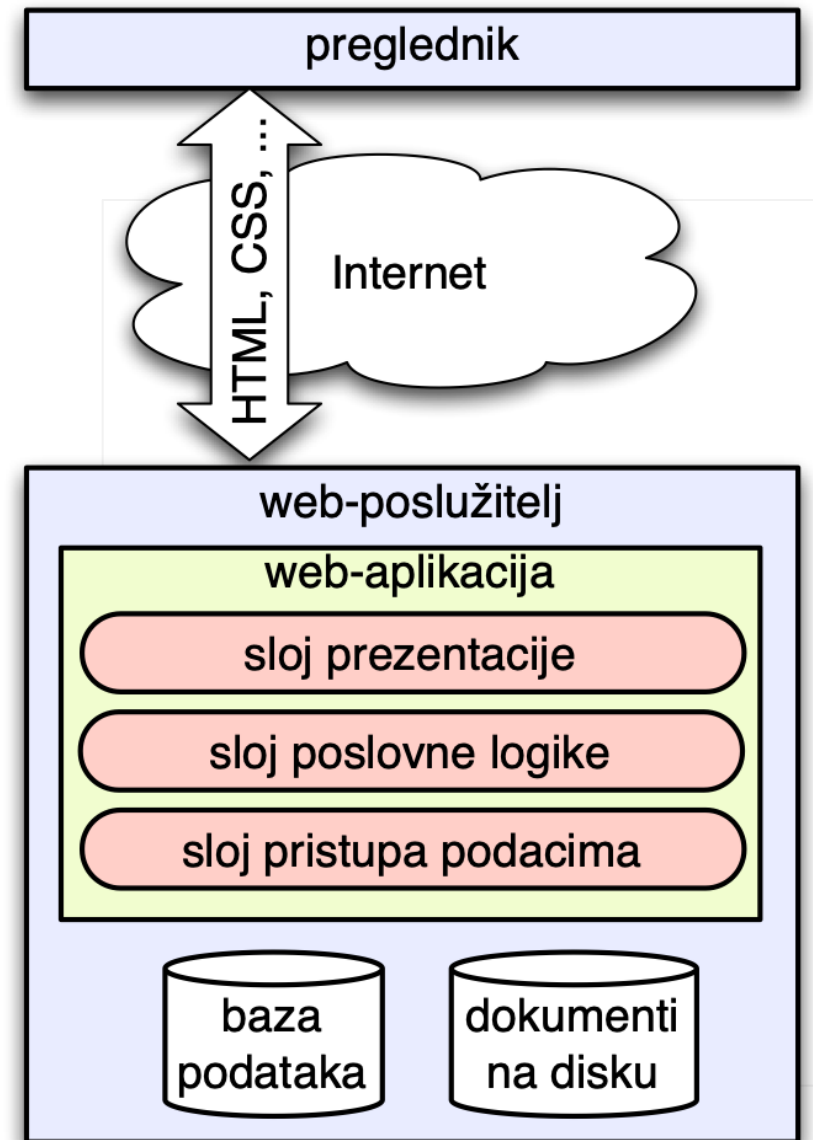
- gumbi za objavljivanje na drugim stranicama (npr. Facebook ili Twitter)





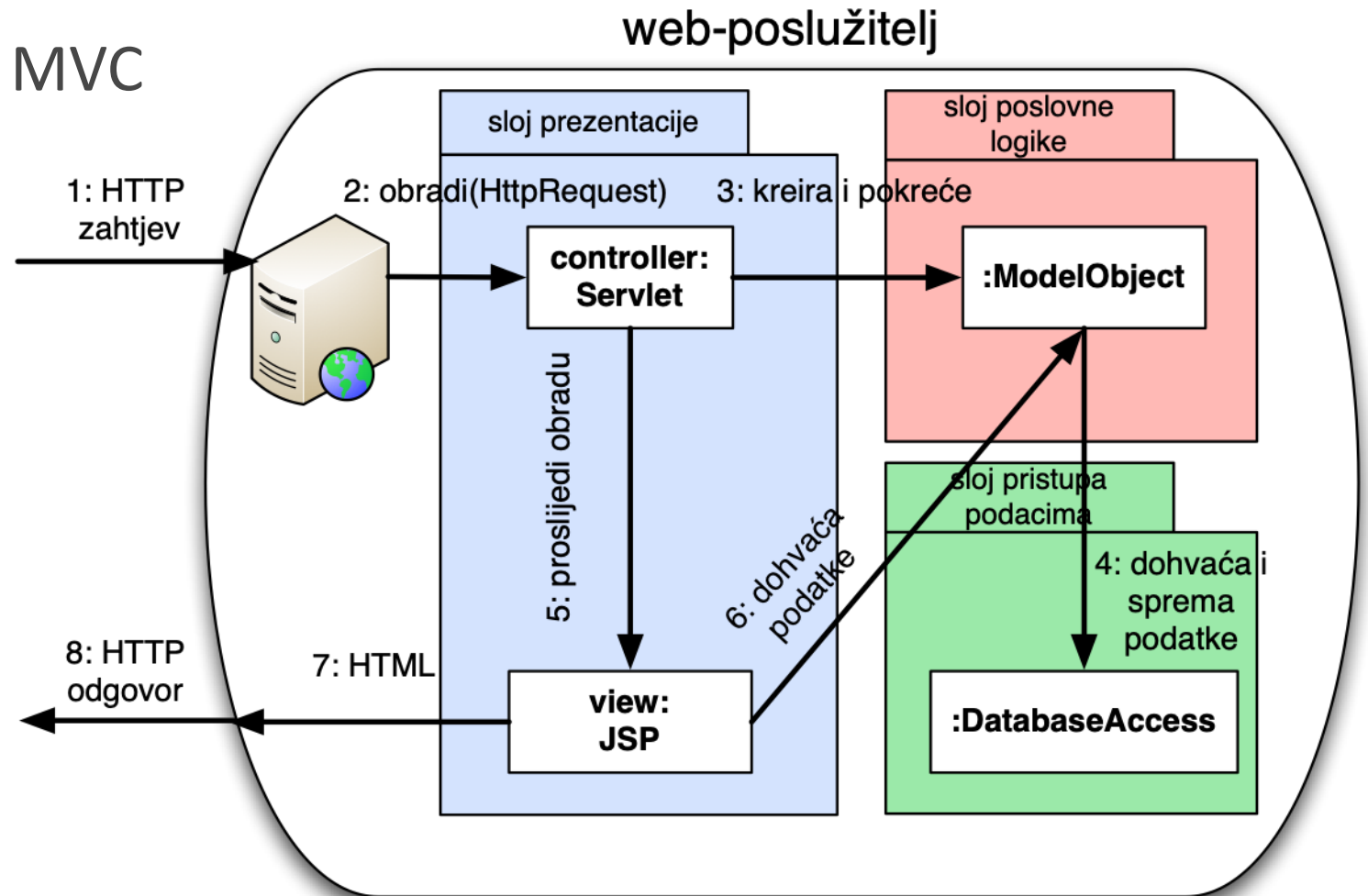
# Arhitektura web-aplikacija

- višeslojna arhitektura
- sloj prezentacije
  - prikaz informacija: GUI, HTML, klikovi mišem, ...
  - obrada HTTP zahtjeva
- sloj poslovne (domenske) logike
  - obrada podataka
  - glavni dio sustava koji radi ono za što je sustav namijenjen
- sloj pristupa podacima
  - komunicira s bazom podataka i drugim komunikacijskim sustavima
  - brine se o transakcijama
  - brine se za pohranu podataka



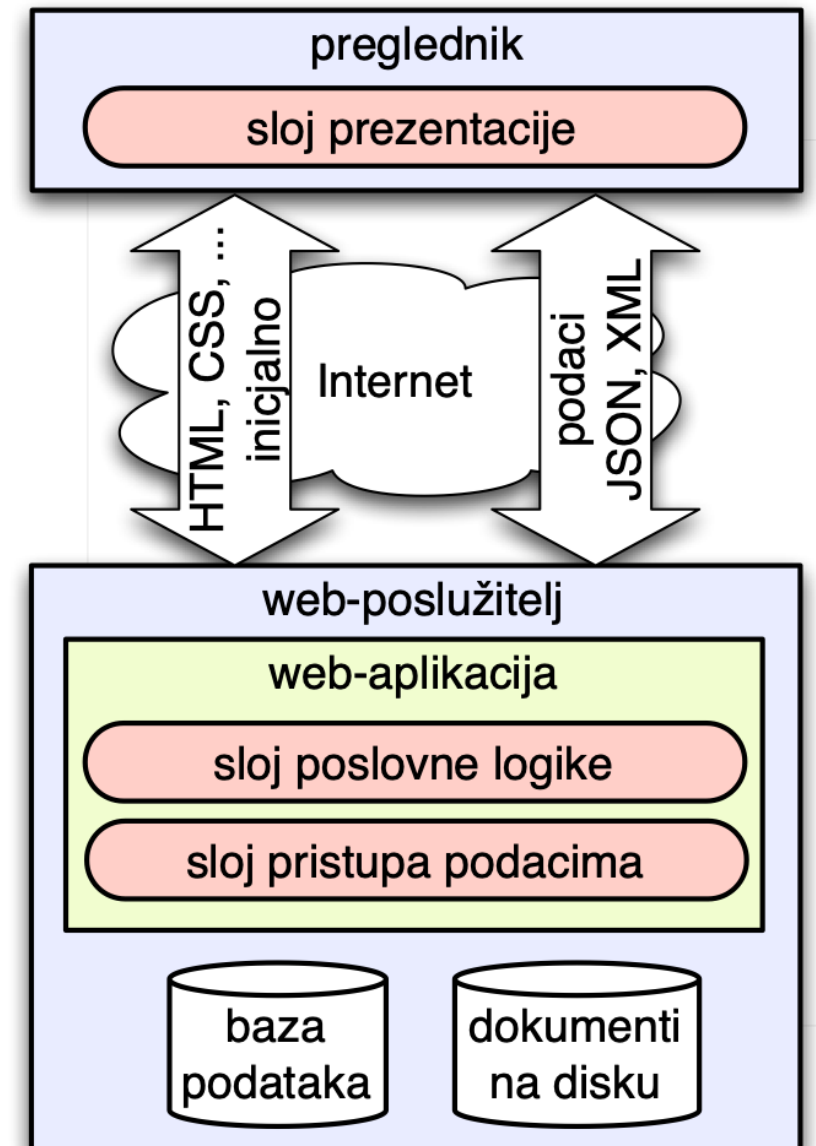
# Arhitektura web-aplikacija - MVC u Javi

- *Model View Controller - MVC*



# Nova arhitektura web-aplikacija

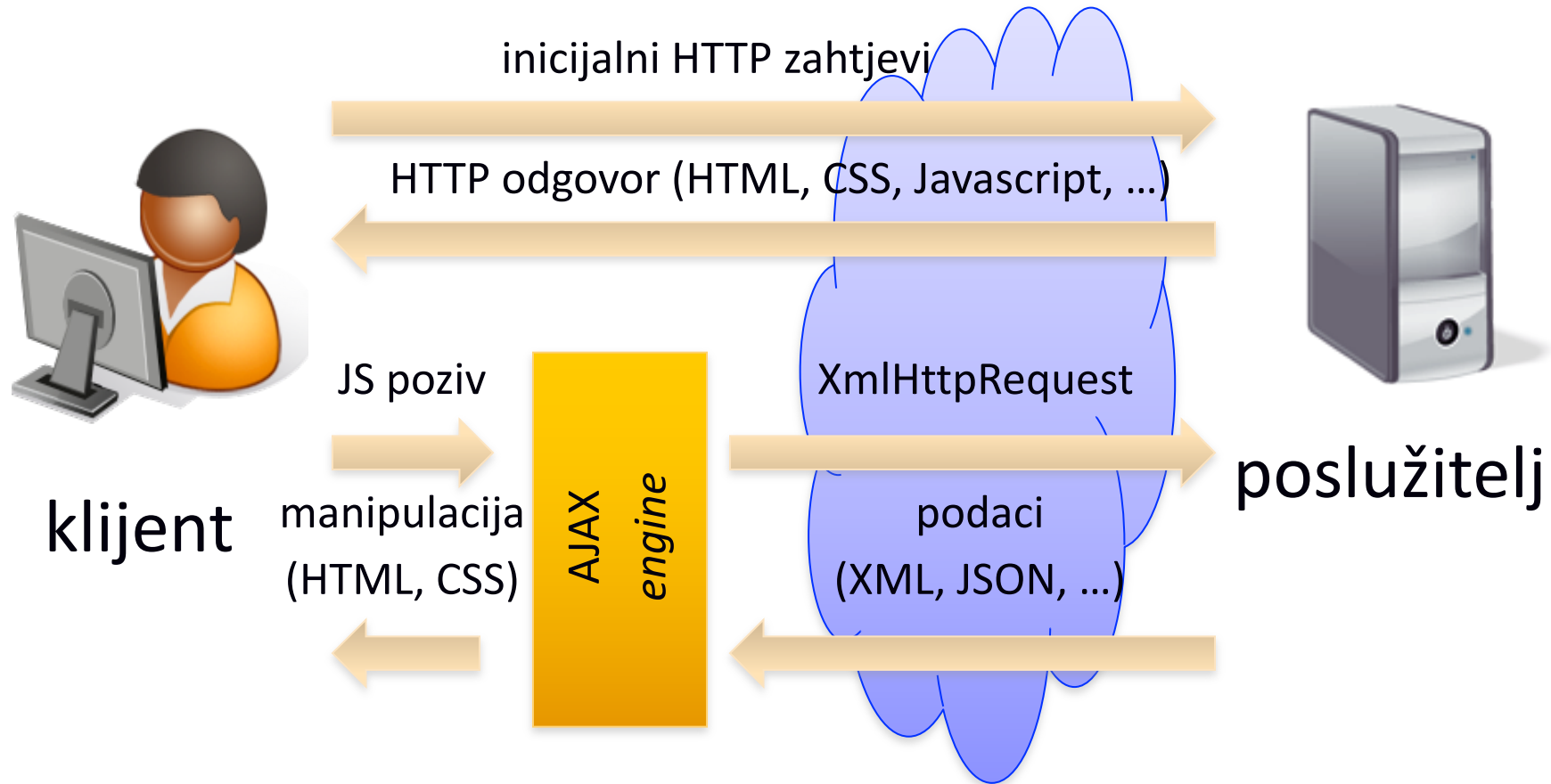
- inicijalno pokretanje
  - HTML, CSS, JavaScript, ...
- za vrijeme korištenja aplikacije
  - podaci putuju u obliku:
    - JSON - JavaScript Object Notation ili
    - XML
  - koristi se AJAX - [Asynchronous JavaScript](#) and XML



# Skripte na klijentu - dio sloja prezentacije

- uključene u HTML ili u posebnoj datoteci koja je povezana
- obično se koristi:
  - **JavaScript** (Netscape), JScript (Microsoft), ECMAScript
- ECMAScript – standardiziran
  - specifikacije ECMA-262 i ISO/IEC 16262
  - svojstva: **dinamički**, **slabo** povezan, objektni, **funkcijski**
  - nema veze s jezikom Java osim imena JavaScript
- svrha:
  - dinamički elementi
  - **interakcija** s korisnikom
  - **provjera obrazaca**
  - komunikacija s poslužiteljem (**AJAX**)
  - ...

# AJAX (Asynchronous JavaScript and XML)



- iz JavaScripta poslati upit na poslužitelj
- odgovor nije nova stranica već podatak (u formatu **JSON**, XML ili ...)
- dobije se na **dinamičnosti** web-stranice

# Poznate knjižnice u JavaScriptu

- popis se stalno mijenja i stalno raste
  - <https://www.javascripting.com>
- DOM manipulation:
  - React (<https://facebook.github.io/react/>)
  - jQuery (<http://jquery.com/>)
  - Preact (<https://preactjs.com>)
- Korisničko sučelje (*User Interface*):
  - Material-UI (<https://material-ui.com>)
  - Ant Design (<https://ant.design>)
  - Code Mirror (<http://codemirror.net>)
- Općenito:
  - Angular (<https://angular.io>)
  - React Native (<http://facebook.github.io/react-native/>)
  - Material-UI (<https://material-ui.com>)
  - Ant Design (<https://ant.design>)

# Web 2.0

- pojam nastao u O'Reilly Media (2003.)
- ideja: Internet kao platforma poput operacijskog sustava
- potiče inovacije i sastavljanje funkcionalnih dijelova iz neovisnih izvora
- korisnici postaju središte
  - interaktivnost, sudjelovanje, objavljivanje
- višemedijski sadržaji (glazba, video, lokacije, slike, karte, ...)

# Usporedba weba 1.0 i 2.0

## Web 1.0

- čitanje informacija
- klijent-poslužitelj
- HTML
- portali
- parsiranje informacija
- posjedovanje
- modemska veza i HW
- direktoriji
- tipična tvrtka: Netscape

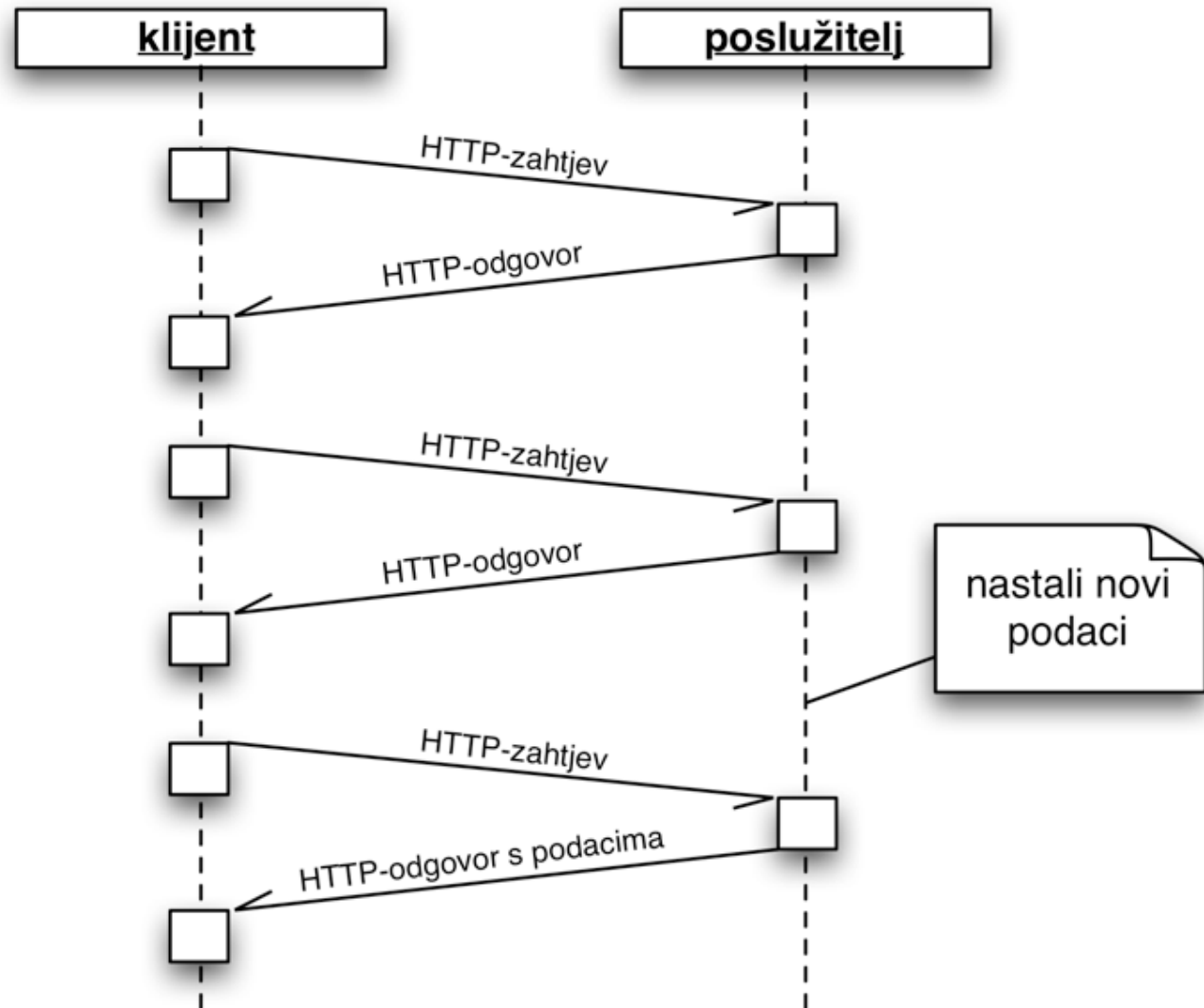
## Web 2.0

- pisanje informacija
- P2P (osobe i aplikacije)
- XML, JSON, HTML5
- usluge (*services*)
- REST API sučelja, RSS
- dijeljenje
- širokopojasna veza i SW
- oznake (*tag*)
- tipične tvrtke: Google, Facebook, Twitter, ...



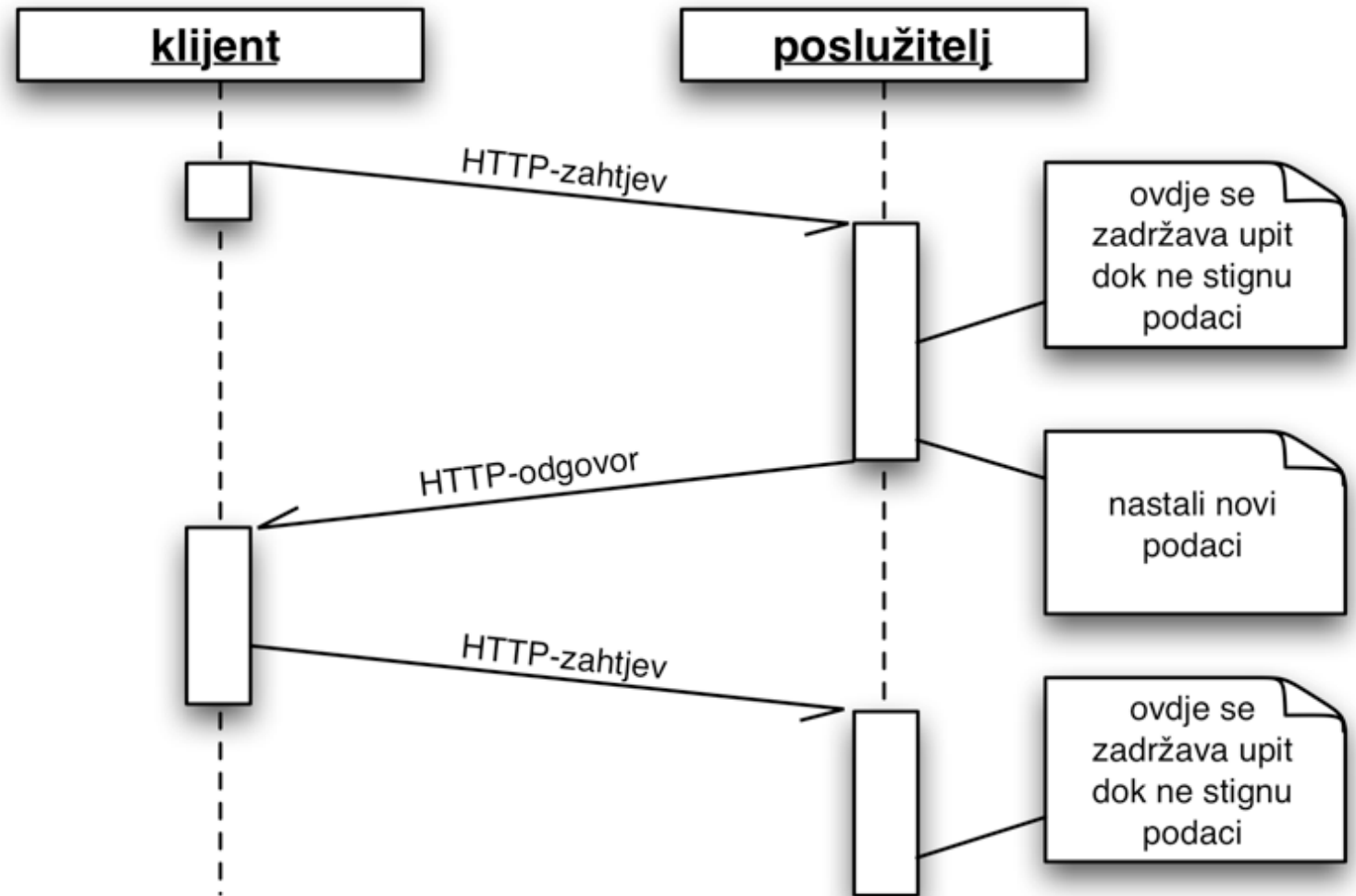
# Slanje događaja klijentu iz preglednika (1) - prozivanje

- prozivanje poslužitelja (*poll*)



# Slanje događaja klijentu iz preglednika (2) – dugo prozivanje

- dugo prozivanje poslužitelja (*long poll*)



# Slanje događaja klijentu iz preglednika (3) - SSE

- **Server-Sent Events** - SSE
- definirano standardom [Server-Sent Events](#) iz 2015.
- omogućuje slanje događaja klijentu kroz otvorenu konekciju
- klijent šalje zaglavlje **Accept: text/event-stream**
- poslužitelj kroz istu konekciju šalje tekst
  - **svaka poruka je odijeljena praznim retkom**
  - poruka ima polja koja su slična zaglavljima HTTP-a:
    - **event** - vrsta događaja (opcionalno)
    - **data** - podaci (obavezno)
    - **id** - identifikator paketa (opcionalno)
    - **retry** - vrijeme ponovnog spajanja u milisekundama (opcionalno)
- [članak - SSE, PHP primjer](#)

Primjer poslanih podataka:

data: Prva poruka

data: Druga poruka

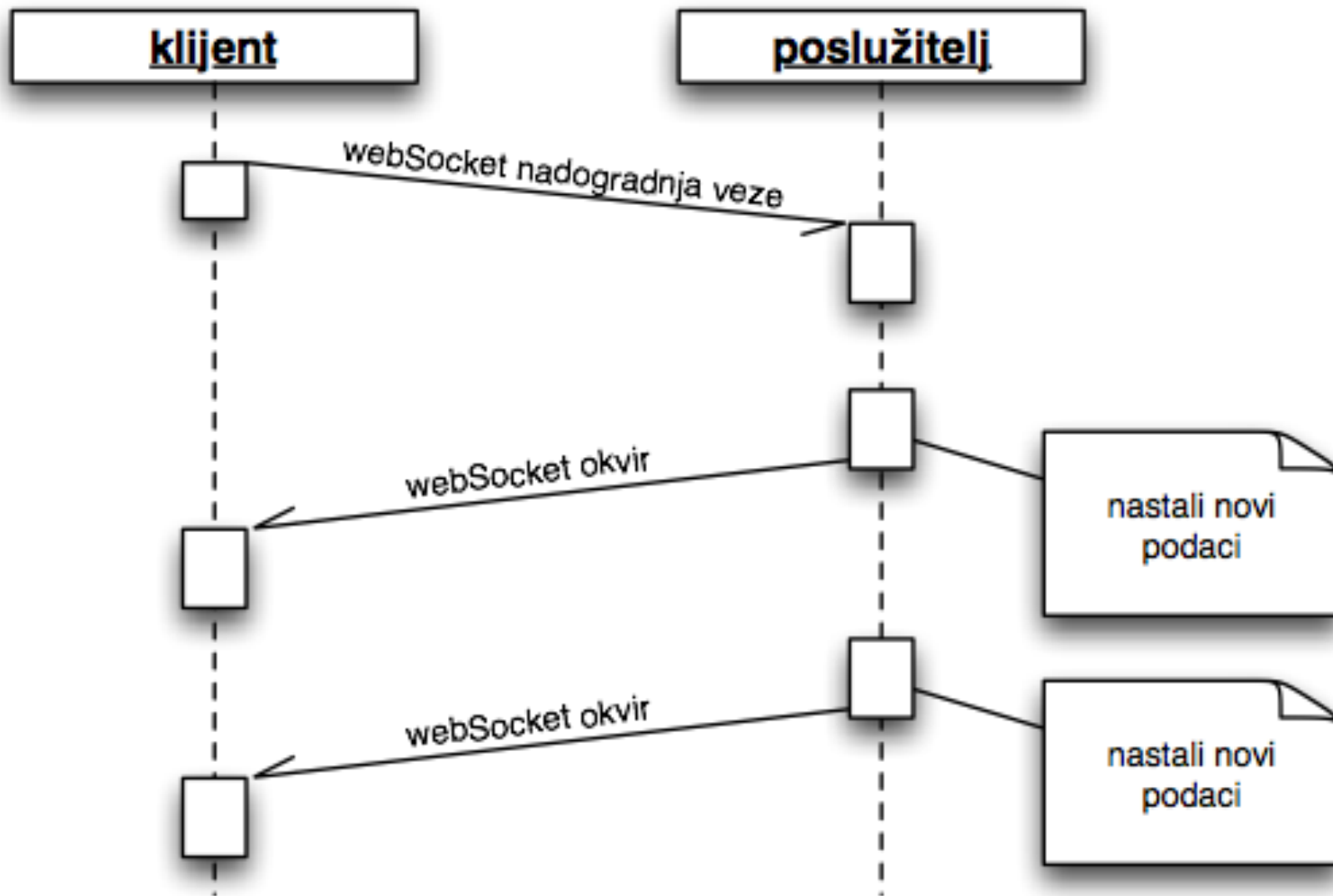
data: 2. red 2. poruke

data: Treća poruka

# Slanje događaja klijentu iz preglednika (4) - **WS**

- The **WebSocket** (WS) Protocol - RFC 6455 - prosinac 2011.
- omogućuje komunikaciju nalik TCP-ovskoj komunikaciji, ali iz internetskih preglednika
- podržavaju full-duplex, istovremeno je moguće primiti i slati podatke
- inicijalna uspostava veze je kompatibilna s HTTP-om da bi isti poslužitelji mogli na istim vratima primiti i HTTP-ovske zahtjeve i Web Socket zahtjeve
- nakon toga slijedi razmjena podataka u okrvirima
- URL shema: ws: (80) ili wss: (443) - ista vrata kao i HTTP
- koristi se: igre, kolaborativne aplikacije, financijske aplikacije, feed na društvenim mrežama, strujanje video sadržaja, ...

# Web Socket - komunikacija

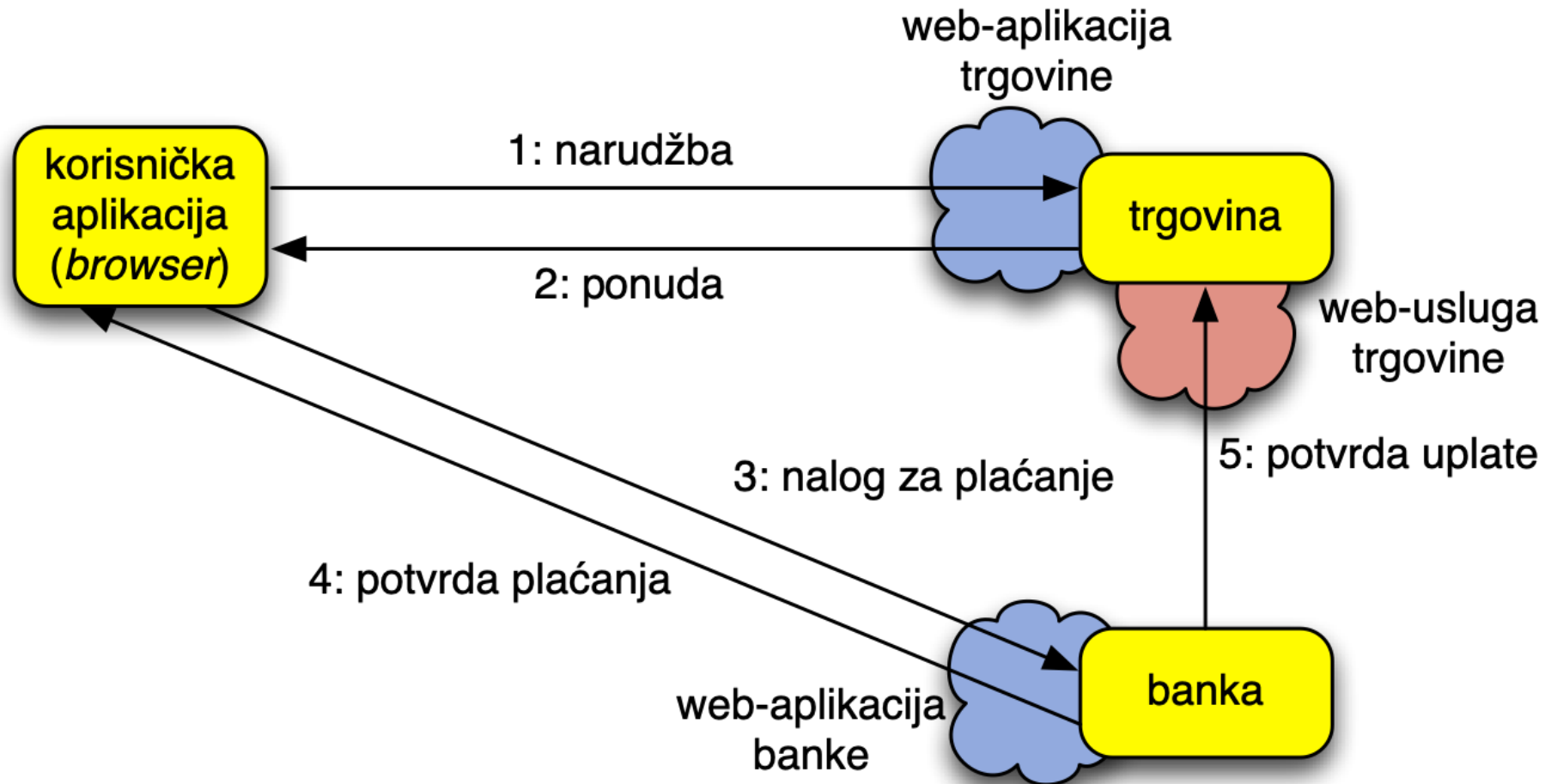


# Web Socket - sadržaj

- 3 vrste sadržaja poruke:
  - tekstualni
  - binarni
  - ping-pong
    - ping-pong služi za provjeru dostupnosti druge strane
      - na poruku ping druga strana mora čim prije odgovoriti porukom pong jednakog sadržaja (maksimalna dužina sadržaja ping i pong poruka je 125 okteta)
      - na više primljenih ping poruka odgovara se samo jednom pong porukom, a samoinicijativnu pong poruku se ignorira
- podprotokoli
  - WebSocket ne definira aplikacijski protokol
  - Previše "posla" za programera kod parsiranja poruka
  - Možemo koristiti aplikacijske protokole preko WebSocketa
    - dogovor prilikom pregovaranja
    - npr. STOMP, WAMP, MQTT, XMPP, ...

# Web-usluge

# Motivacija: scenarij kupovine





# Uvod u web-usluge

- “obični” Web tipično koriste ljudi za pribavljanje informacija
- tehnologija Web Services (WS) omogućuje programima lakše korištenje Web  
  - umjesto RPC-a/Java RMI-a, CORBA-e, DCOM-a, itd.
- svojstva:
  - komunikacija se temelji na XML-u
  - koristi već postojeću internetsku infrastrukturu i protokole
  - usluge dostupne putem Interneta
  - omogućuje integraciju između različitih aplikacija
  - ne ovisi o programskom jeziku ili zatvorenoj tehnologiji jedne tvrtke
  - omogućuje otkrivanje usluga koje nude te aplikacije
  - usluge su slabo povezane
  - temelji se na industrijskim standardima

# Što je web-usluga?

«Web-usluga je aplikacija identificirana URI-jem, čija se sučelja i veze mogu definirati, opisati i otkriti XML-om i koja podržava direktnu interakciju s drugim aplikacijama putem internetskih protokola koristeći poruke temeljene na XML-u»

([www.w3.org](http://www.w3.org))

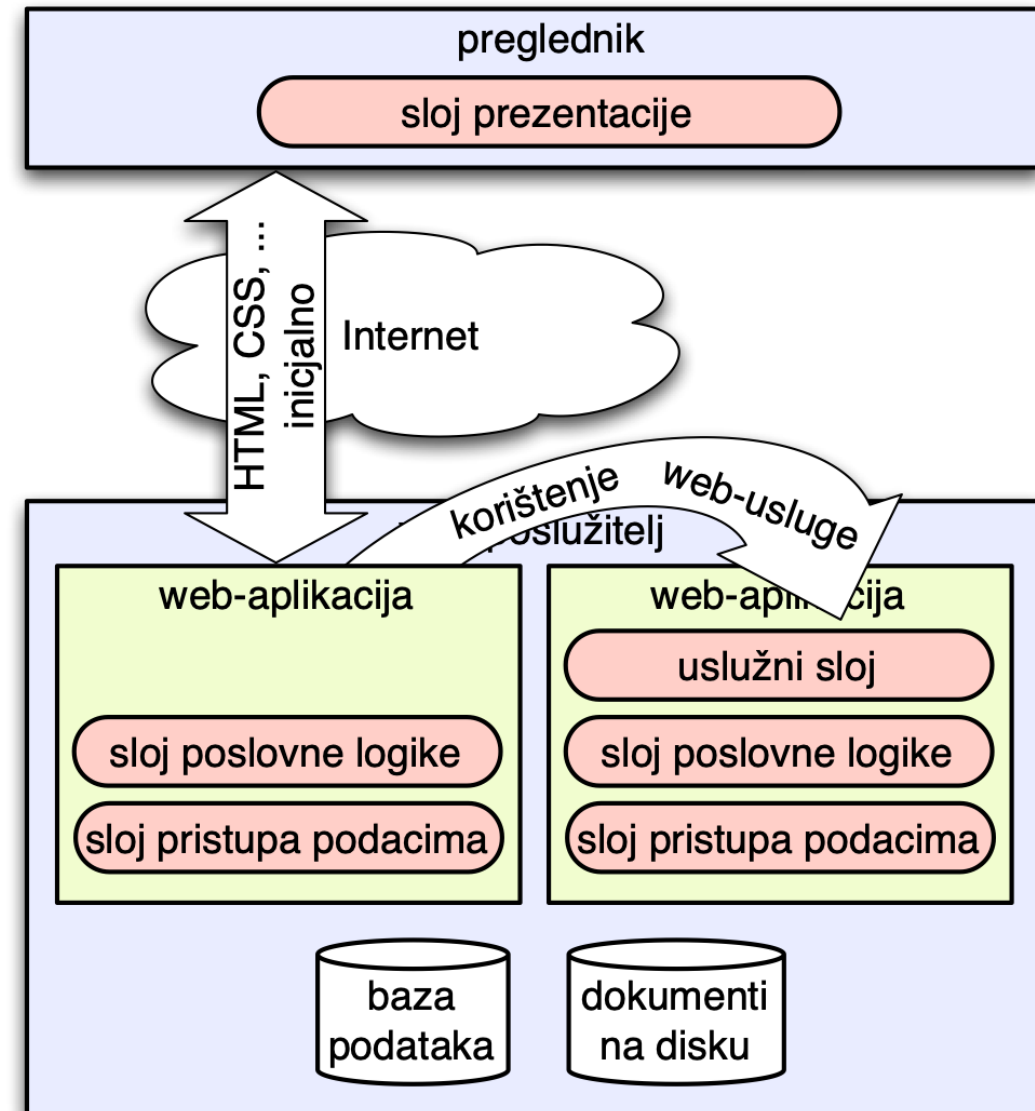
«Tehnologija web-usluga predstavlja novu vrstu web-aplikacija. To su samostalne, samoopisujuće aplikacije građene od modula, a koje se mogu objaviti, otkriti i pobuditi putem Weba. Web-usluge obavljaju funkcije koje mogu biti bilo što, od jednostavnih zahtijeva do kompliciranih poslovnih transakcija...»

(IBM's tutorial, [www.xml.com](http://www.xml.com))

- web-usluga je program koji:
  - je identificiran URI-jem
  - komunicira s klijentskim programima putem Weba
  - ima sučelje (API) opisano standardima web-usluga
  - omogućuje korištenje neovisno o platformama i programskim jezicima

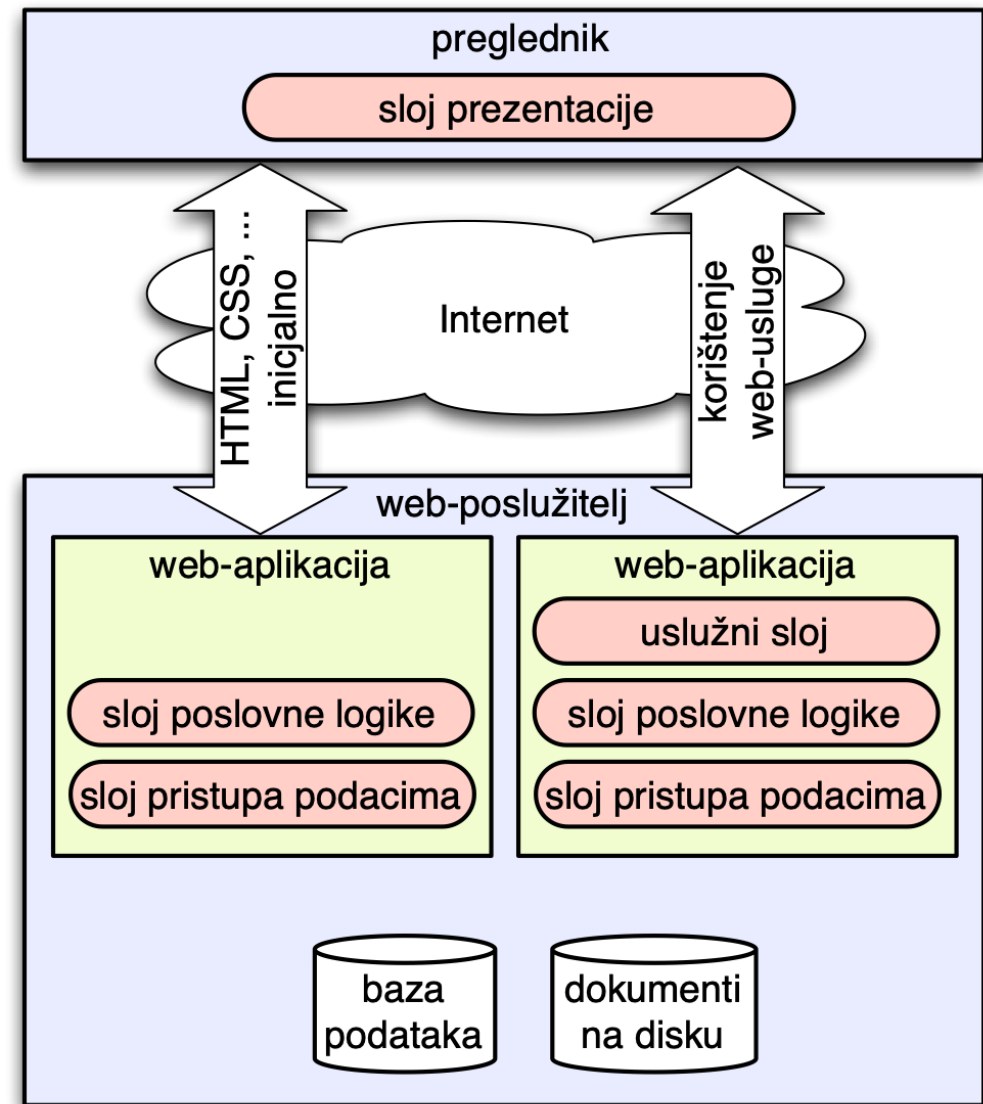
# Web-aplikacija koja koristi web-usluge (1)

- web-aplikacija koristi web-uslugu
- web-usluga ne mora biti na istom računalu (u principu nije)



# Web-aplikacija koja koristi web-usluge (2)

- klijent koristi web-uslugu (npr. klijent je aplikacija na pametnom telefonu ili preglednik koji koristi JavaScript za korištenje usluge)
- ovdje se obično radi o web-usluzi koja se temelji na REST-u



# Vrste web-usluga

- **Usluge temeljene na udaljenim procedurama (RPC)**
  - rade kao poziv udaljenih procedura
  - koriste protokol SOAP i specifikaciju WSDL
  - na poslužitelju se poziva metoda u objektu
  - podaci su povezani s metodama koje se pozivaju
- **Usluge temeljene na dokumentima/porukama**
  - definiraju se poruke koje se razmjenjuju (XML Schema, WSDL)
  - koriste protokol SOAP i specifikaciju WSDL
  - nisu jako povezane
  - nije bitna implementacija, već samo podaci
- **Usluge temeljene na prijenosu prikaza stanja resursa (REST)**
  - RESTful (*Representational state transfer*) ili REST-usluge
  - temelje se na protokolu HTTP
  - koriste metode protokola: GET, PUT, DELETE, POST, PATCH

# Tehnologije

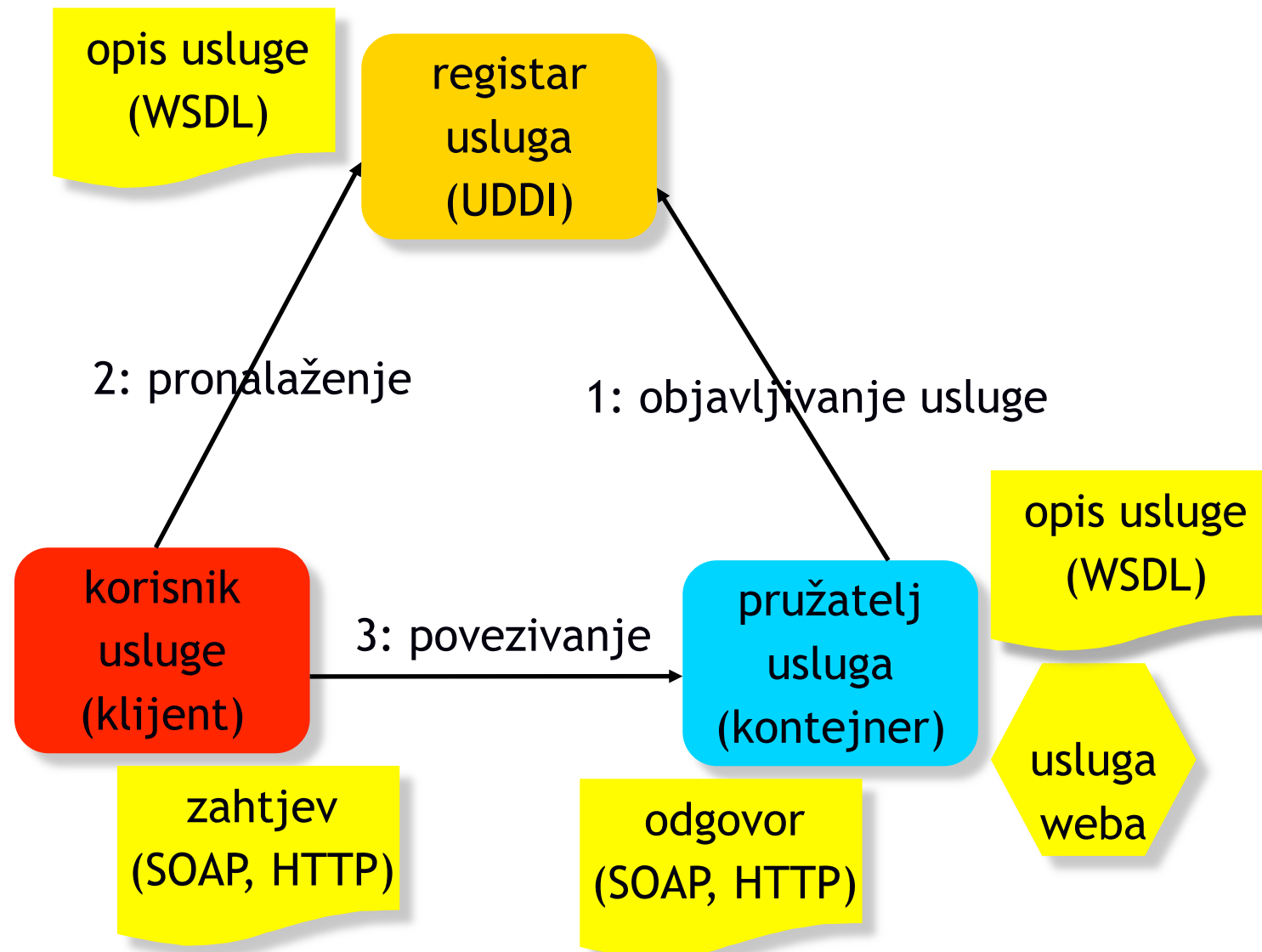
## Tehnologije web-usluga

- WSDL (*Web Services Definition Language*)
  - opisuje uslugu
- SOAP (*Simple Object Access Protocol*)
  - format poruke
- UDDI (*Universal Description, Discovery and Integration*)
  - za otkrivanje usluga

## Druga generacija web-usluga (WS-\*)

- WS-Coordination
  - protokol za koordinaciju distribuiranih aplikacija
- WS-Transaction
- BPEL4WS (*Business Process Execution Language*)
  - jezik za formalnu specifikaciju poslovnih procesa i interakcijskih protokola
- WS-Security
  - sigurnosni protokol (TLS, integritet, privatnost, ...)
- WS-ReliableMessaging
- WS-Policy
- WS-Attachments
- WS-Addressing
  - adresiranje usluga i poruka

# Arhitektura i korištenje

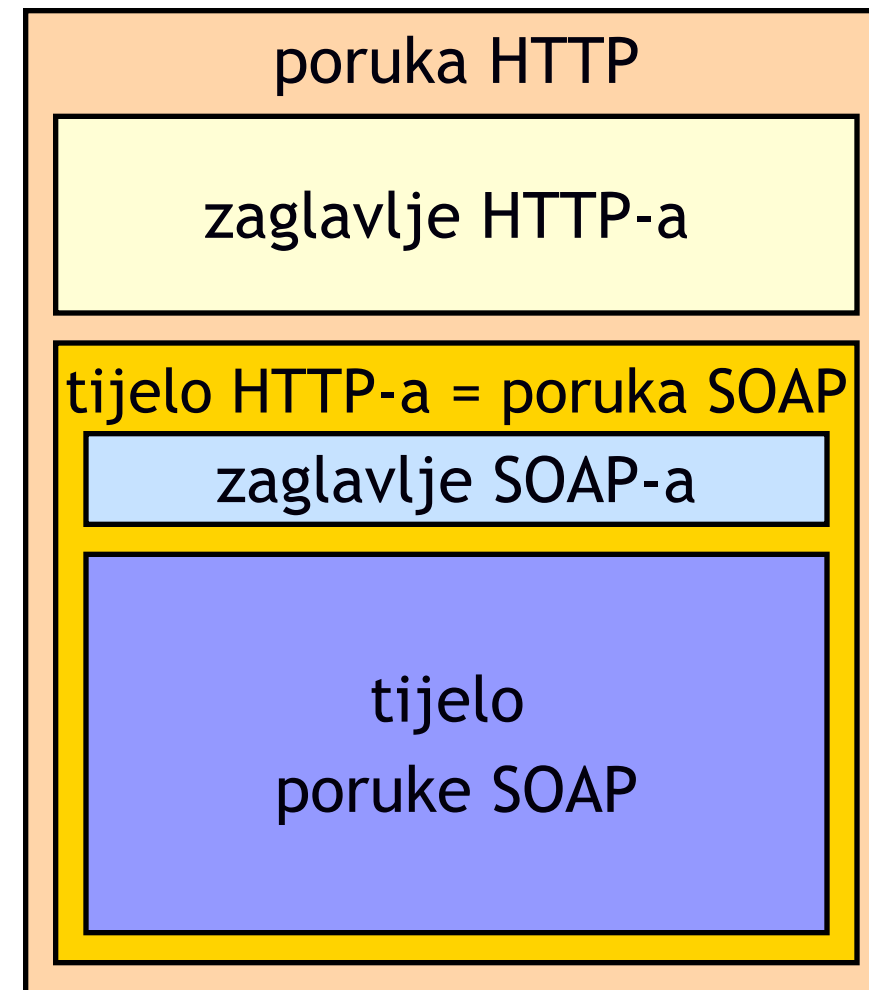
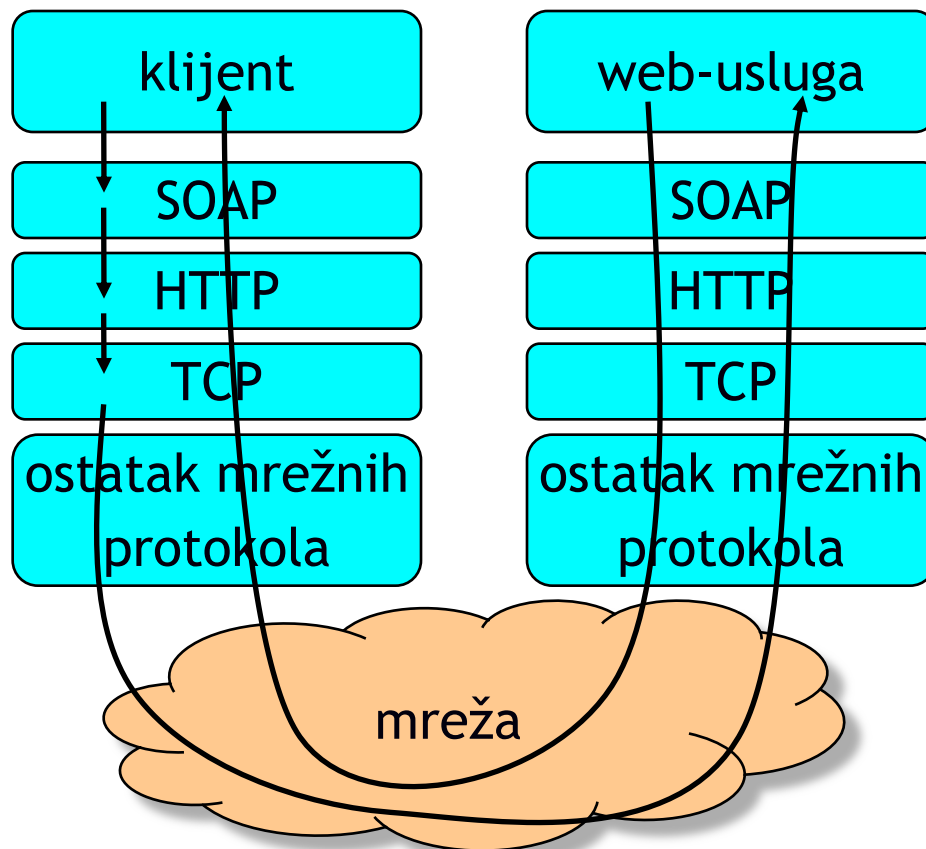


# SOAP (*Simple Object Access Protocol*)

- omogućuje komunikaciju s web-uslugom
- dva osnovna načina rada:
  - poziv udaljenih procedura (RPC)
    - slično kao Corba, DCOM, Java RMI
    - služi za prijenos serijaliziranih parametara i rezultata
    - posljedica:
      - dobro definirana sučelja i tipovi podataka
      - prilagodni kod može biti generiran automatski
  - razmjena dokumenata/poruka
    - sadrži XML dokument
    - fleksibilnije u odnosu na RPC
    - XSLT i XQuery se koristi za prilagodbu dokumenata
    - lakše se koriste uzorci razmjene poruka
    - polako se uključuje semantički web (ontologije, procesiranje i sl.)
- specifikacija: <http://www.w3.org/TR/soap/>



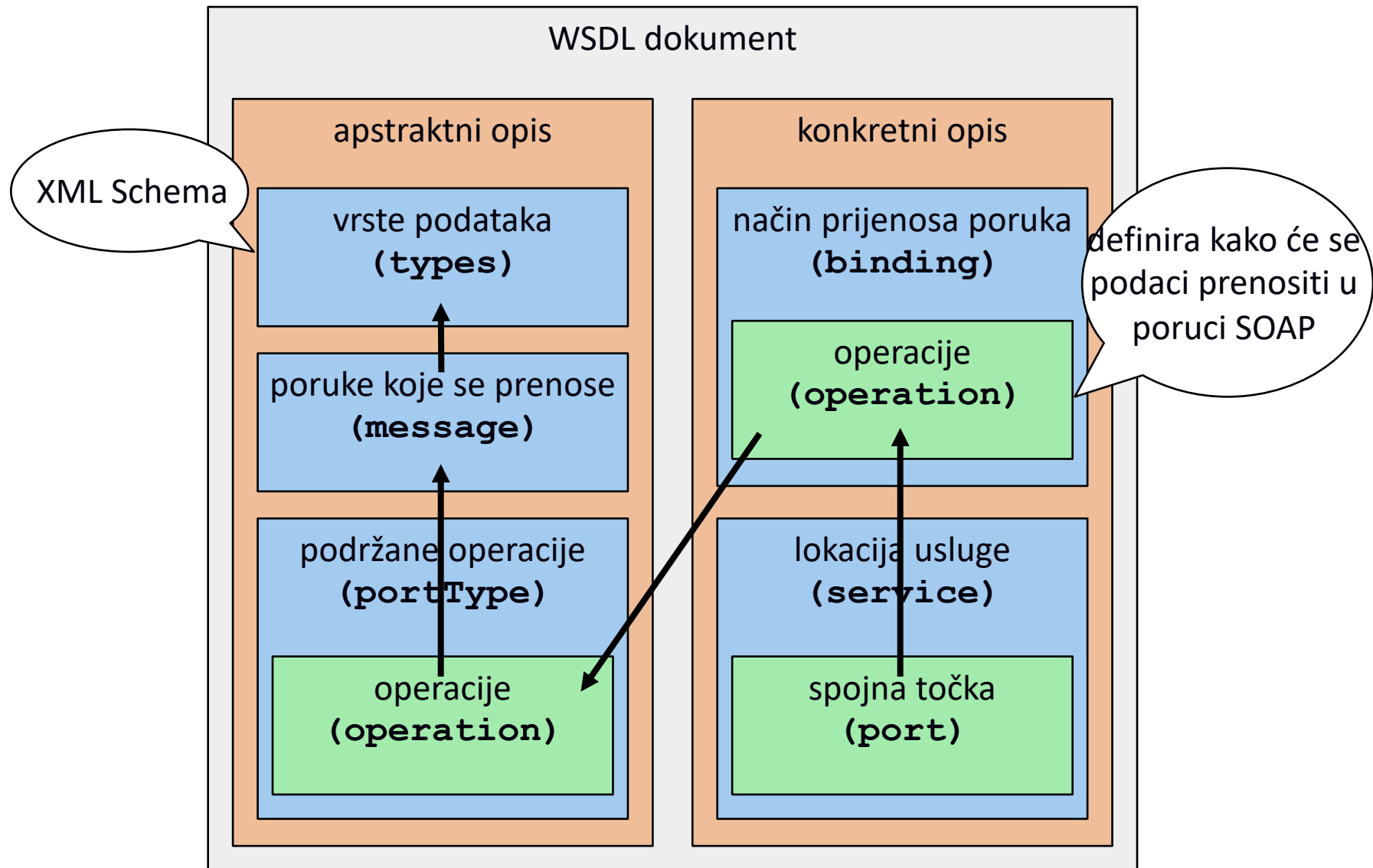
# Prijenos poruke SOAP



# WSDL (Web Services Description Language)

- jezik za opis web-usluga
- web-usluga je opisana skupom komunikacijskih krajnjih točaka (*ports*)
- krajnja točka se sastoji od dva dijela:
  - apstraktne definicije operacija i poruka
  - specifikacije mrežnog protokola i pojedine krajnje točke te formata poruke
- opisuje komunikacijske detalje između klijenta i usluge
  - strojevi (računala) mogu pročitati WSDL
  - mogu pozvati uslugu definiranu WSDL-om
- jedan je od mehanizama koji omogućuje da se usluga može otkriti pomoću registra
- specifikacija: <https://www.w3.org/TR/wsd1>

# Struktura WSDL-a



# Elementi WSDL-a (1)

- **definitions**
  - osnovni element WSDL dokumenta
- **types**
  - opis podataka pomoću XML Scheme
  - nepotreban ako se koriste osnovni podaci iz Scheme
- **message**
  - opis jednosmjerne poruke
  - definira ime poruke
  - poruka se sastoji od dijelova (**part**)
  - svaki dio se referencira na vrste podataka iz dijela **types**
- **portType**
  - definira operacije
  - svaka operacija se sastoji od poruka (reference na poruke)
  - poruke koji mogu biti:
    - parametri (ulazne poruke)
    - vrijednosti koje se vraćaju (rezultati)

# Elementi WSDL-a (2)

- **binding**
  - definira kako će se poruke iz operacije prenositi
  - definira koje **transportne protokole** će koristiti (HTTP GET, HTTP POST, SOAP, SMTP)
  - stil definira vrstu čitave poruke:
    - **rpc** - zahtjev će imati omotač u kojem će pisati naziv funkcije koja se poziva
    - **document** - zahtjev i odgovori će imati “obične” XML dokumente
  - poruke mogu koristiti dvije vrste pakiranja poruka:
    - **literal** - onako kako definira Schema
    - **encoded** - kodirano pravilima u SOAP-u
- **services**
  - definira lokaciju usluge (URL)
- **import**
  - koristi se za uključivanje drugih WSDL ili XML Schema dokumenata

# Pronalaženje usluga

- UDDI (*Universal Description, Discovery and Integration*)
- osigurava platformu za otkrivanje usluga na Internetu
- sastoji se od 3 dijela:
  - imenik (*White pages*)
    - adrese, kontakti i identifikatori
  - poslovni imenik (*Yellow pages*)
    - kategorizacija područja, usluga i proizvoda te lokacije
  - tehničke informacije (*Green pages*)
    - sadrži tehničke informacije o uslugama
- nema centralnog registra
- više informacija: <http://uddi.xml.org/uddi-org>
- standard: <https://www.oasis-open.org/standards#uddiv3.0.2>

# Web-usluge temeljene na RPC-u

- napravimo novi projekt (npr. NetBeans)
- stvorimo novu web-uslugu i stvorimo operacije/metode kao u primjeru dolje
- alat sam generira WSDL i iz WSDL-a klijenta

```
@WebService()  
public class MyService {  
    @WebMethod(operationName = "add")  
    public int add(@WebParam(name = "x") int x,  
                  @WebParam(name = "y") int y)  
    {  
        return x+y;  
    }  
  
    @WebMethod(operationName = "toLowerCase")  
    public String toLowerCase(  
        @WebParam(name = "text")  
        String text)  
    {  
        return text.toLowerCase();  
    }  
}
```

# Usluge temeljene na dokumentima

- Usluge temeljene na dokumentima
  - razmjenjuju se dokumenti
  - dogovor o dokumentima (XML Schema)
  - obično su asinkrone
  - parametri kod povezivanja u WSDL-u su: Document-literal
- Postupak izrade:
  1. definiranje XML Scheme dokumenata
    - obično u posebnoj datoteci
  2. definiranje WSDL-a
    - uključuje XML Scheme
  3. iz WSDL-a se mogu generirati:
    - kostur usluge
    - primer klijenta



# Web-usluge temeljene prijenosu prikaza stanja resursa (REST)

- REST (*Representational State Transfer*)
- pojmovi: *The REST Way* ili *RESTful services*
- pojam je skovao Roy Fielding u svojoj doktorskoj disertaciji
- nije standard već arhitekturni stil
- sve se temelji na resursima koji su predstavljeni URL-ovima:
  - `http://localhost:8080/RassusRest/rest/persons` - popis svih korisnika
  - `http://localhost:8080/RassusRest/rest/persons/1` - korisnik s identifikatorom 1
- koristi protokol: HTTP (GET, POST, PUT, DELETE, PATCH)
- koristi podatke: JSON, XML, sirovi (npr. za slike, video)
- bez stanja (*stateless*), priručni spremnik (*cache*)

# Format JSON (Javascript Object Notation)

- podatak - par: **ime**/**vrijednost**

**"firstName" : "Ivana"**

- vrijednosti mogu biti:

- broj, *string*, *boolean* (true, false), polje, objekt, null

- objekt su parovi u vitičastim zagradama, npr.:

```
{ "firstName" : "Ivana", "lastName" : "Podnar Žarko" }
```

- ime u objektu mora biti jedinstveno

- polje su vrijednosti u uglatim zagradama, npr.:

```
[ { "name": "Ignac Lovrek", ... },  
  { "name": "Ivana Podnar Žarko", ... },  
  ... ]
```

# Primjer podataka u JSON-u i XML-u

- JSON:

```
{ "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "room" : "C7-12",  
  "phone" : "261",  
  "_links" : {  
    "self" : {  
      "href" : http://localhost:8080/persons/2  
    }  
  }  
}
```

- XML:

```
<person>  
  <firstName>Ivana</firstName>  
  <lastName>Podnar Žarko</lastName>  
  <room>C7-12</room>  
  <phone>261</phone>  
  <links>  
    <link>  
      <rel>self</rel>  
      <href>http://localhost:8080/persons/2</href>  
    </link>  
  </links>  
</person>
```

# Usporedba JSON-a i XML-a

- JSON

- za
  - format
  - ugrađen u preglednike
- protiv
  - nema ugrađene poveznice (koristi se kao tekst)
  - nema mogućnosti definiranja sheme
  - ograničen skup vrsta podataka
  - problem binarnih podataka

- XML

- za
  - mogu se zapisati složene strukture podataka
  - ima standardne poveznice
  - dobar skup alata za transformaciju i obradu
- protiv
  - problem binarnih podataka
  - loš za strukture bez redoslijeda
  - ograničenja DTD-a
  - postoji puno standarda koji ga kompliciraju
  - brzina procesiranja

# Primjer zahtjeva i odgovora (1)

HTTP GET <http://localhost:8080/persons>

```
[  
  {  
    "id" : 1,  
    "name" : "Ignac Lovrek",  
  },  
  {  
    "id" : 2,  
    "name" : "Ivana Podnar Žarko",  
  },  
  {  
    "id" : 3,  
    "name" : "Mario Kušek",  
  },  
  {  
    "id" : 4,  
    "name" : "Krešimir Pripužić",  
  },  
  ...  
]
```

# Primjer zahtjeva i odgovora (2)

HTTP GET <http://localhost:8080/persons/2>

```
{  
  "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "phone" : "261",  
  "room" : "C7-12"  
}
```

# Svojstva metoda protokola HTTP

- Svojstva:
  - Sigurna (*safe*) - bez posljedica za podatke
  - *Idempotentna* - može se izvršavati više puta
  - Može se privremeno spremiti odgovor (*cacheable*)
- Metode:
  - GET - sigurna, idempotentna, može se privremeno spremiti
  - PUT - idempotentna
  - DELETE - idempotentna
  - HEAD - sigurna, idempotentna
  - POST - ništa
  - PATCH - idempotentna

# Tipično korišćenje usluga REST

- za *Create, Read, Update and Delete* (CRUD)

HTTP	CRUD
POST	Create, (Overwrite/Replace)
GET	Read
PUT	Update, (Create, Delete)
DELETE	Delete
PATCH	Partial update

- primjer gotove usluge: *Twitter*



# Dizajniranje usluge

- paziti na mrežu:
  - što i koliko se podataka prenosi
  - koliko zahtjeva i odgovora imamo da bismo nešto prikazali na klijentu
- napraviti URL za svaki resurs
- definirati metode za svaki resurs
- stavljati poveznice u resursima
  - ne vraćati čitavu strukturu
- specificirati format za svaki resurs
- povezati usluge tako da sve kreće preko jednog URL-a
  - HATEOAS - *Hypermedia as the Engine of Application State* (3. razina zrelosti web-usluga)
- preko vrsta medija dogovarati verzije usluga (API-ja)

# Zadatak

- Napraviti uslugu koja služi za evidenciju plaćenih računa
  - U sustavu imamo osobe
  - Svaka osoba ima osobne podatke
  - Osobe se mogu stvoriti, obrisati i promijeniti im podatke
  - Svaka osoba ima račune koje je platila za pojedini mjesec
  - Kada je neki račun unesen više ga nije moguće mijenjati niti obrisati
- 
- Rješenje projekta i klijenata koji ga koriste se nalazi na:  
<https://gitlab.tel.fer.hr/spring>

# Tablica dizajna usluge

resurs	metode	šalje	svrha
/persons	POST	osoba	stvara novu osobu
	GET		vraća popis osoba
/persons/{id}	GET		vraća osobu s ID-om
	DELETE		briše osobu (brišu se i svi računi te osobe)
	PUT	osoba	mijenja podatke o osobi
	PATCH	dio osobe	mijenja samo podatka koji su poslani
/persons/{id}/bills	POST	račun	stvara novi račun za osobu s ID-om
	GET		vraća popis računa te osobe
/bills/{bid}	GET		vraća račun s ID-om bid

# Primjer upita i odgovora - stvaranje resursa

- POST /persons
- sadržaj zahtjeva:

```
{  
  "firstName": "Jura",  
  "lastName": "Jurić",  
  "address": "Unska 3, 10000 Zagreb",  
  "phone": "+385 1 6129 999",  
  "email": "jura.juric@fer.hr"  
}
```

- odgovor:

201 Created

Location: <http://localhost:8080/persons/4>

# Primjer upita i odgovora - lista osoba

- GET /persons

- odgovor:

200 OK

```
[  
  {  
    "id" : 1,  
    "name" : "Ignac Lovrek",  
  },  
  {  
    "id" : 2,  
    "name" : "Ivana Podnar Žarko",  
  },  
  {  
    "id" : 3,  
    "name" : "Mario Kušek",  
  },  
  {  
    "id" : 4,  
    "name" : "Krešimir Pripužić",  
  },  
  ...  
]
```

# Primjer upita i odgovora - dohvaćanje resursa

- GET /person/2

- odgovor:

```
{  
  "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "address": "Unska 3, 10000 Zagreb",  
  "phone": "+385 1 6129 999",  
  "email": "ivana.podnar@fer.hr"  
}
```

# Primjer upita i odgovora - dohvaćanje nepostojećeg

- GET /person/100
- odgovor:

404 Not Found

# Primjer upita i odgovora - promjena resursa

- PUT /person/2

```
{  
  "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "address": "Unska 3, 10000 Zagreb",  
  "phone" : "+385 1 6129 761",  
  "email": "ivana.podnar@fer.hr"  
}
```

- odgovor:


204 No Content



# Primjer upita i odgovora - stvaranje resursa

- PUT /person/200

```
{  
  "firstName" : "Ana",  
  "lastName" : "Anić",  
  "address": "Unska 3, 10000 Zagreb",  
  "phone": "+385 1 6129 999",  
  "email": "ana.anic@fer.hr"  
}
```



200 se  
ignorira

- odgovor:

201 Created

Location: <http://localhost:8080/persons/5>

# Primjer upita i odgovora - promjena dijela resursa

- PATCH /person/2

```
{  
  "phone" : "+385 1 6129 761"  
}
```

- odgovor:

200 OK

```
{  
  "firstName" : "Ivana",  
  "lastName" : "Podnar Žarko",  
  "address": "Unska 3, 10000 Zagreb",  
  "phone" : "+385 1 6129 761",  
  "email": "ivana.podnar@fer.hr"  
}
```

# Primjer upita i odgovora - promjena nepostojećeg resursa

- PATCH /person/200

```
{  
  "phone" : "+385 1 6129 761"  
}
```

- odgovor:

404 Not Found

# Primjer upita i odgovora - osoba koja postoji

- DELETE /person/3
- odgovor:

204 No Content

# Primjer upita i odgovora - osoba koja ne postoji

- DELETE /person/300
- odgovor:

404 Not Found

# Primjer upita i odgovora - stvaranje resursa

- POST /persons/2/bills

- sadržaj zahtjeva:

```
{  
  "month": 3,  
  "year": 2019,  
  "amount": 250  
}
```

- odgovor:

201 Created

Location: <http://localhost:8080/bills/1>

# Primjer upita i odgovora - lista računa

- GET /persons/2/bills
- odgovor:

200 OK

```
[  
  {  
    "id": 1,  
    "peroid": "2019-3"  
  },  
  {  
    "id": 3,  
    "peroid": "2019-4"  
  },  
  ...  
]
```

# Primjer upita i odgovora - jedan račun

- GET /bills/1
- odgovor:

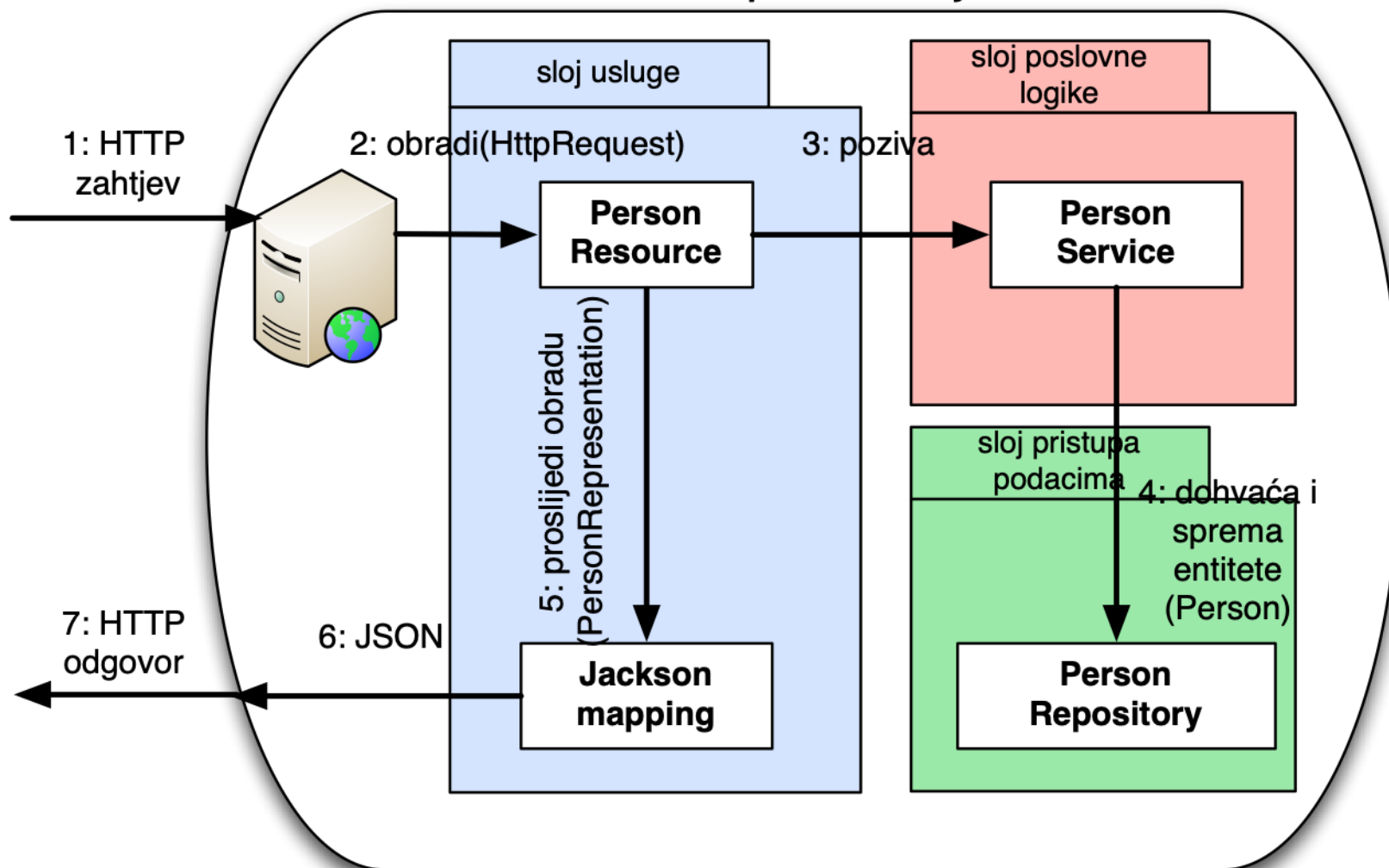
200 OK

```
{  
  "month": 3,  
  "year": 2019,  
  "amount": 250  
}
```



# Arhitektura web-aplikacije (REST)

web-poslužitelj



# Spring Framework - <http://spring.io>



- prva verzija 2003. koju je napravio Rod Johnson
- radni okvir za lakši razvoj aplikacija
- bavi se konfiguracijom objekata u sustavu (IoC – *inversion of control*)
  - upravlja poslovnim objektima kao običnim objektima (POJO – *plain old java objects*)
  - brine se za kreiranje objekata
  - povezuje kreirane objekte (IoC, *wiring up*, DI - *dependency injection*)
  - upravlja njihovim životnim ciklusom
- složene veze između objekata se definiraju u XML-u ili pomoću bilješki (*annotation*)
- odvaja poslovnu logiku od mehanizama za ispravan rad sustava (transakcije, logiranje, ...)
- vrlo je složen za početnika jer ima puno stvari ugrađeno

# Spring Boot (trenutna verzija 2.3.4)

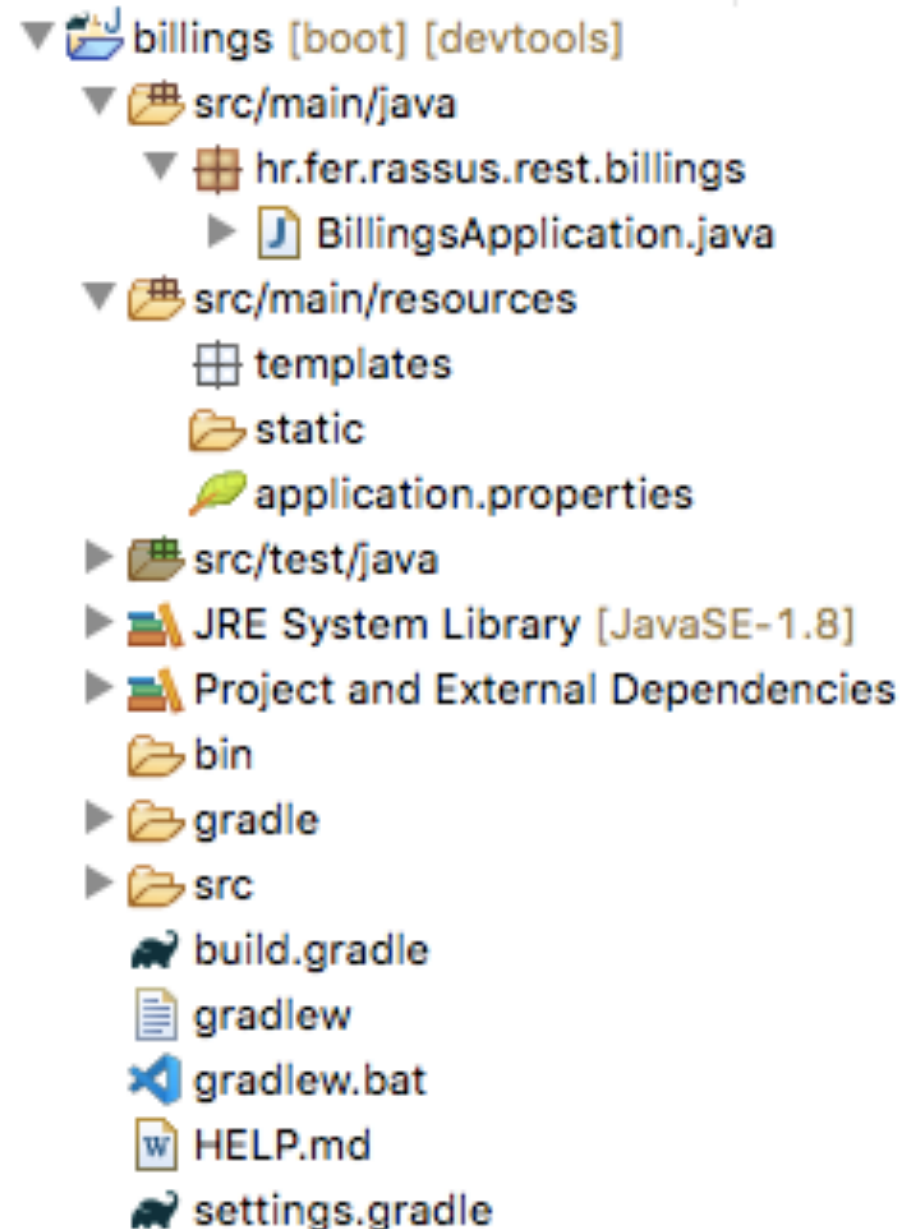
- jedan od podprojekata Springa
  - <http://projects.spring.io/spring-boot/>
- pojednostavnjuje korištenje Springa, pogotovo stvaranje novog projekta
- podržava:
  - automatsku konfiguraciju
  - pretraživanja klasa na putu (*path*)
- aplikacija ima manje koda
- kod web-aplikacija - omogućuje izradu samostalnih aplikacija
  - web-poslužitelj zapakiran u jar
  - jednostavnije instaliranje
  - aplikacija spremna za produkcijsku okolinu
- [primjeri projekata](#), [Spring framework guru tutorials](#)
- **Video materijali:**
  - Spring predavanja za preddiplomski projekt - [youtube](#),
  - Vještina RUAZOSA - meduza ([1. dio](#), [2. dio](#))

# Stvaranje Spring projekta

- otvoriti stranicu <https://start.spring.io>
- klik na "Switch to the full version"
- popuniti:
  - Group: hr.fer.rassus.rest
  - Artifact: billings
  - Name: billings
  - Packaging: Jar
  - Java Version: 11 (može i novija verzija)
  - Language: Java
- odabrati: Web, HATEOAS, DevTools, (Lombok?)
- klik na Generate Project - napravi zip koji preglednik skine
- trebamo u IDE-u otvoriti projekt u koji smo raspakirali

# Struktura projekta

- pogledati:
  - BillingsApplication
    - klasa koja se pokreće
  - build.gradle
    - skripta za "građenje"
  - application.properties
    - vanjska konfiguracija



# klasa LecturesApplication

```
@SpringBootApplication
public class BillingsApplication {

    // metoda koja sve pokreće
    public static void main(String[] args) {
        SpringApplication.run(BillingsApplication.class, args);
    }
}
```

# Beanovi i DI (*Dependency Injection*)

- Objekte koje stvara i s kojima upravlja Springov kontejner zovu se *Springovi beanovi* (skraćeno samo *bean*)
- DI - *dependency injection*
  - objekti definiraju ovisnosti o drugim objektima
  - mora imati ili: atribut, setere ili konstruktor kroz koji se ovisnosti postavljaju
  - kontejner onda ubacuje ovisnosti kada stvara baenove
  - nepotrebne posebne metode za instanciranje i postavljanje ovisnih objekata da bi objekti normalno funkcionirali
- doseg beanova - `@Scope("tip")`
  - **singleton (podrazumijevano)** - jedna instanca u JVM-u
  - prototip - novi objekt svaki put kada se zahtjeva bean
  - zahtjev (*request*) - kod svakog zahtjeva se stvara novi bean
  - sjednica (*session*) - za svakog korisnika jedan bean
  - globalna sjednica - koristi se kod portleta

# Ubrizgavanje beanova

1. preko atributa (*field*) - ne preporuča se

```
@Autowired  
private NekiBean b;
```

2. preko konstruktora

```
private NekiBean b;  
  
public XApplication(NekiBean b) {  
    this.b = b;  
}
```

3. preko setera

```
private NekiBean b;  
  
@Autowired  
public void setB(NekiBean b) {  
    this.b = b;  
}
```



# Definiranje beanova

- dva načina:
  - u konfiguracijskoj klasi (npr. SimpleApplication) definirati metodu koja je označena da vraća bean (@Bean)
  - označiti klasu s @Component
    - mehanizam pretraživanja puta može pronaći takvu klasu
    - podvrste:
      - **@Controller** - predstavlja kontroler u Web MVC-u
      - **@Service** - predstavlja namjeru da je to usluga
      - **@Repository** - predstavlja komponentu koja pristupa podacima
      - **@RestController** - predstavlja kontroler u Web MVC-u koja vraća podatke koji se serializiraju u JSON ili XML

# REST kontroler - *bean*

```
@RestController
public class PersonResourceController {

    @GetMapping("/persons")
    public String getPersonsList() {
        return "Pero i Ana";
    }
}
```

# Primjer dohvaćanja

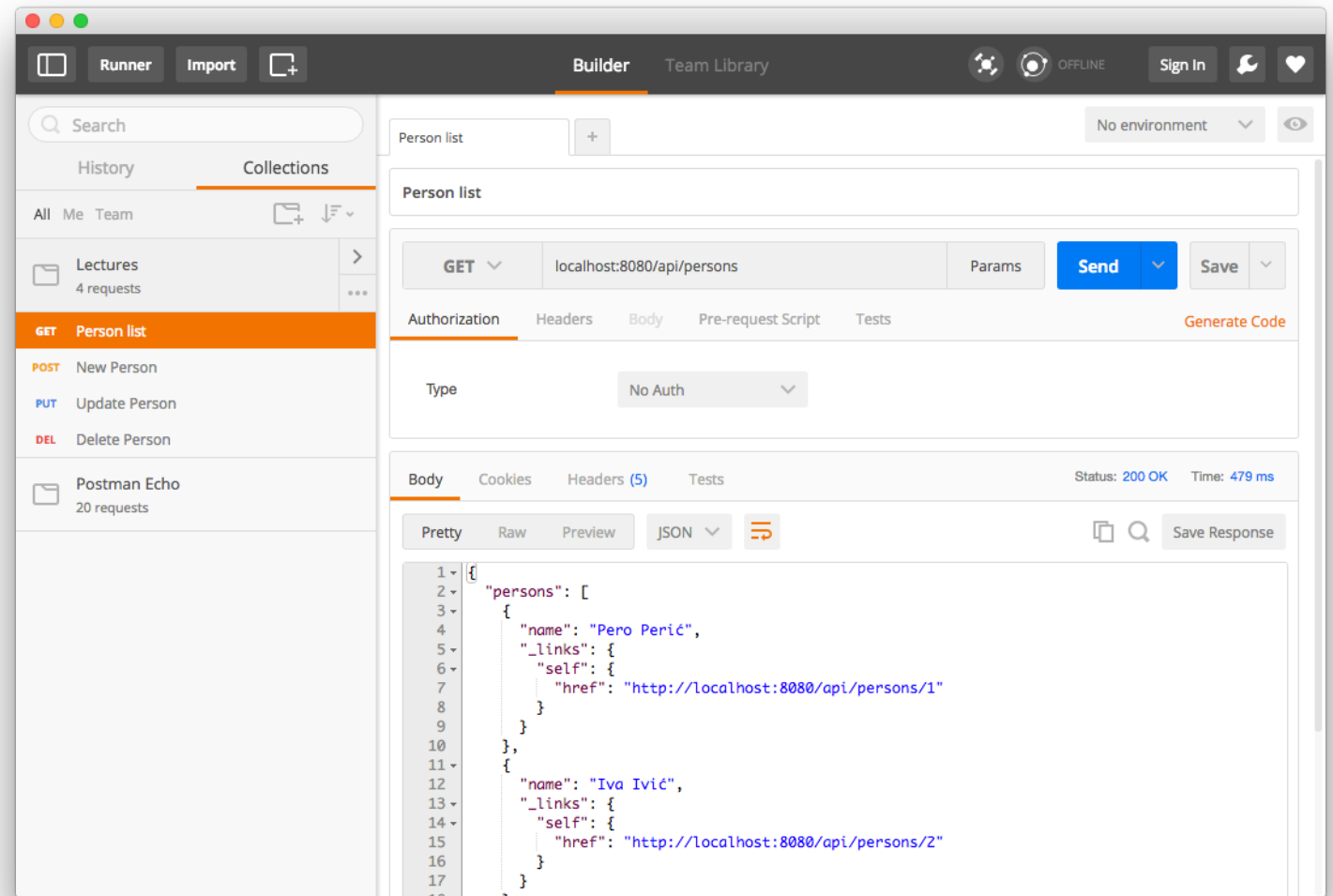
- Chrome extensions

(<https://chrome.google.com/webstore/search/rest>):

- Advanced Rest Client ili
- [Postman](#)

- u terminalu:

- [curl](#)
- [HTTPIe](#)



# REST kontroler - *bean* (2)

```
@RestController
public class PersonResourceController {

    @GetMapping("/persons")
    public String getPersonsList() {
        return "Pero i Ana";
    }

    @GetMapping("/persons/{id}")
    public String getPerson(@PathParam("id") String id) {
        return "Ana";
    }
}
```

# REST kontroler - *bean* (3)

```
@RestController
public class PersonResourceController {

    @GetMapping("/persons")
    public String getPersonsList() {
        return "Pero i Ana";
    }

    @GetMapping("/persons/{id}")
    public PersonRepresentation getPerson(@PathParam("id") String id) {
        return new PersonRepresentation("Ana", "Anić",
            "Unska 3, 10000 Zagreb", "+385 1 6129 999", "ana.anic@fer.hr");
    }
}

public class PersonRepresentation {
    private String firstName, lastName, address, phone, email;

    // getters, setters, konstruktori (prazan, atributi), toString
}
```

# PersonService

```
@Service
public class PersonService {

    private int pidCounter = 0;
    private Map<Integer, Person> persons = new HashMap<>();

    public Collection<Person> getPersons() {
        return persons.values();
    }
}
```

# PersonResourceController

```
@RestController
public class PersonResourceController {

    private PersonService personService;

    public PersonResourceController(PersonService personService) {
        this.personService = personService;
    }

    @GetMapping("/persons")
    public Collection<ShortPersonRepresentation> getPersonsList() {
        return personService.getPersons().stream()
            .map(p -> PersonAssembler.toShortPersonRepresentation(p))
            .collect(Collectors.toList());
    }

    ...
}
```

# ShortPersonRepresentation

```
public class ShortPersonRepresentation {  
    private int id;  
    private String name;  
  
    // setters/getters/konstruktori  
}
```



# PersonAssembler

```
public class PersonAssembler {  
  
    public static ShortPersonRepresentation toShortPersonRepresentation(  
        Person person) {  
        return new ShortPersonRepresentation(  
            person.getId(),  
            person.getFirstName() + " " + person.getLastName());  
        }  
  
    public static PersonRepresentation toPersonRepresentation(Person person)  
    { ... }  
  
    public static Person toPerson(PersonRepresentation personRepresentation)  
    { ... }  
  
    public static Person toPerson(int id,  
        PersonRepresentation personRepresentation) { ... }  
  
    public static void updatePersonForNotNullValues(Person person,  
        PersonRepresentation personRepresentation) { ... }  
}
```

# Dohvaćanje jedne osobe

```
@RestController
public class PersonResourceController {
    ...
    @GetMapping("/persons/{id}")
    public ResponseEntity<PersonRepresentation> getPerson(
        @PathVariable("id") Integer id)
    {
        Person person = personService.getPerson(id);
        if(person != null) {
            return ResponseEntity.ok(
                PersonAssembler.toPersonRepresentation(person));
        }
        return ResponseEntity.notFound().build();
    }
}
```

# Kreiranje nove osobe

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
...
@RestController
@RequestMapping("/persons")
public class PersonResourceController {
    ...
    @PostMapping()
    public ResponseEntity<?> newPerson(
        @RequestBody PersonRepresentation personRepresentation) {
        Person person = PersonAssembler.toPerson(personRepresentation);
        personService.newPerson(person);

        return ResponseEntity
            .noContent()
            .location(linkTo(
                methodOn(this.getClass()).getPerson(person.getId())).toUri())
            .build();
    }
    ...
}
```

# Model zrelosti web-usluga

- model je napravio Leonard Richardson
  - <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>
- Razine:
  - 0
    - pozivanje usluga je većinom pozivanje udaljenih procedura
    - jedan URI jedna HTTP metoda (većinom XML-RPC i SOAP)
  - 1
    - koriste više resursa, ali su nazivi metoda i parametara enkodirani u URL
    - više URI-ja, jedna HTTP metoda (GET)
  - 2
    - koriste više resursa i HTTP metoda te statusne kodove
    - ne koriste svoju shemu (vrste podataka koji se prenose)
    - primjer Amazon S3
  - 3
    - isto kao razina 2, ali koristi hipermedijske vrste
    - resurs opisuje svoje mogućnosti i veze

# Standardi, prijedlozi ...

- URI Template - ožujak 2012., IETF, [RFC 6570](#)
  - standard za ekspanziju varijabli iz URI-ja
- JSON Hypertext Application Language (HAL) - svibanj 2016., IETF, v08
  - [draft-kelly-json-hal-08](#)
  - prijedlog višemedijskih tipova za reprezentaciju resursa i njihovih relacija
- Application-Level Profile Semantics (ALPS) - kolovoz, 2015., IETF, v02
  - [draft-amundsen-richardson-foster-alps-02](#)
  - format podataka za opis aplikacije i njihove semantike
- [OpenAPI inicijativa](#) (konzorcij)
  - osnovana 2016. u sklopu Linux Foundation
  - OpenAPI specification (bivši Swagger 2.0 spec.) - aktualna verzija [3.0.3](#)



# Pitanja za ponavljanje

- Koje su dvije vrste web-aplikacija i koja je razlika između njih?
- Kakva je arhitektura web-aplikacije i svrha svakog sloja?
- Koja je razlika između nove i stare arhitekture web-aplikacija?
- Čemu služe skripte na klijentu?
- Što je i kako radi AJAX?
- Usporedite dva načina slanja događaja s poslužitelja (web-aplikacije) klijentu (preglednik).
- Što su i čemu služe web-usluge?
- Objasnite dva načina kako web-aplikacija koristi web-usluge.
- Koja je razlika između Weba 1.0 i Weba 2.0?
- Koje su 3 vrste web-usluga i razlike između njih?

# Pitanja za ponavljanje

- Što je SOAP i čemu služi?
- Što je WSDL i čemu služi?
- Kakva je struktura WSDL-a u čemu služe pojedini elementi?
- Što je UDDI i čemu služi?
- Koja je razlika između web-usluga RPC i temeljenih na dokumentima?
- Kakve su web-usluge temeljene prijenosu prikaza stanja resursa i što ih karakterizira?
- Koje su razlike između JSON-a i XML-a?
- Koja su svojstva metoda protokola HTTP te ih objasnite?
- Objasnite model zrelosti web-usluga.
- Na što treba paziti kod dizajniranja REST usluge?
- Objasniti postupak dizajniranja REST usluge?