

```

<h1>Map Example</h1>
<div id="ti">
    <label for="start-time">Pocetno vrijeme:</label>
    <input type="datetime-local" id="start-time" name="start-time"> <br/>
    <label for="end-time">Završno vrijeme:</label>
    <input type="datetime-local" id="end-time" name="end-time"><br/>
    <button style="margin: 5px" onclick="filterByTime()">Filtriraj</button>
</div>

<div id="map"></div>
<div id="map"></div>

<script src="https://maps.googleapis.com/maps/api/js?
key=AIzaSyDbMvcxG7stkGQNMj6MbIXbUQ3IGTM6dl4&callback=initMap" async defer></script>
<script>
    var coordinates;
    var path;
    var map;
    function initMap() {
        // Kreirajte mapu

        // Dohvatite koordinate sa servera
        fetch('http://localhost:8000/cords')
            .then(response => response.json())
            .then(data => {
                // Definirajte listu GPS koordinata

                coordinates = data.map(item => ({ lat: item.lat, lng: item.lng, time: new Date(item.time * 1000) }));
                var count = coordinates.length;
                var divisions = 2; // Specify the number you want to divide by

                var stepSize = Math.ceil(count / divisions);

                document.getElementById('start-time').value = coordinates[0].time.toISOString().slice(0, 16);
                document.getElementById('end-time').value = new
Date(coordinates[parseInt(coordinates.length)-1].time).toISOString().slice(0, 16);

                // Sort coordinates by time
                coordinates.sort(function(a, b) {
                    return a.time - b.time;
                });

                var first = coordinates[stepSize];
                map = new google.maps.Map(document.getElementById('map'), {
                    center: {lat: first.lat, lng: first.lng}, // Postavite centar mape
                    zoom: 3.5 // Postavite povećanje mape
                });

                // Kreirajte putanju koristeći koordinate
                path = new google.maps.Polyline({
                    path: coordinates,
                    geodesic: true,
                    strokeColor: '#b91674', // Postavite boju putanje
                    strokeOpacity: 2.0,
                    strokeWeight: 5 // Debljina linije putanje
                });
            });
    }
</script>

```

```
});

// Prikazivanje putanje na mapi
path.setMap(map);
})
.catch(error => {
  var errorElement = document.getElementById('error');
  errorElement.textContent = 'Error: Server not working';
  errorElement.style.display = 'block';
  console.error('Error: Server not working', error);
});
}

function filterByTime() {
  var startTime = new Date(document.getElementById('start-time').value);
  var endTime = new Date(document.getElementById('end-time').value);

  // Filtriraj GPS koordinate temeljem odabranog vremenskog raspona
  var filteredCoordinates = coordinates.filter(function(coord) {
    var a = coord.time.getTime() >= startTime.getTime();
    var b = coord.time.getTime() <= endTime.getTime();

    return a && b;
  });

  path.setMap(null);

  path = new google.maps.Polyline({
    path: filteredCoordinates,
    geodesic: true,
    strokeColor: '#0c15af', // Postavite boju putanje
    strokeOpacity: 2.0,
    strokeWeight: 5 // Debljina linije putanje
  });

  path.setMap(map);
}

</script>
</body>
</html>
```

```

/*
 * Copyright 2022 Krešimir Pripučić
 */
package hr.fer.zkist.ppks.persons.httpserver;

import com.sun.net.httpserver.Headers;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;
import hr.fer.zkist.ppks.persons.Cordinates;
import hr.fer.zkist.ppks.persons.CordinatesService;

import java.io.*;
import java.net.InetSocketAddress;
import java.net.URI;
import java.nio.charset.StandardCharsets;

/**
 *
 * @author Krešimir Pripučić <kresimir.pripuzic@fer.hr>
 */
public class PersonsHTTPGetServer {

    private final CordinatesService cordinatesService;
    private final HttpServer httpServer;

    // todo hosted url http://localhost:8000/cords
    public PersonsHTTPGetServer(int port) throws IOException {
        cordinatesService = new CordinatesService();
        httpServer = HttpServer.create(new InetSocketAddress(port), 0);
        httpServer.createContext("/cords", new PersonsHTTPGetServer.PersonsHandler()); // todo change cords
    }

    public void start() {
        httpServer.start();
    }

    public static void main(String[] args) throws IOException {
        PersonsHTTPGetServer server = new PersonsHTTPGetServer(8000);
        server.start();
    }

    class PersonsHandler implements HttpHandler {

        @Override
        public void handle(HttpExchange httpExchange) throws IOException {

            // Add CORS headers
            Headers headers = httpExchange.getResponseHeaders();
            headers.add("Access-Control-Allow-Origin", "*"); // Allow requests from any origin todo add cors
            headers.add("Access-Control-Allow-Methods", "GET"); // Allow GET requests
            headers.add("Access-Control-Allow-Headers", "Content-Type");
        }
    }
}

```

```

headers.add("Content-Type", "application/json; charset=" + StandardCharsets.UTF_8.name());

if (httpExchange.getRequestMethod().equalsIgnoreCase("GET")) {
    // Process the GET request and return the coordinates

    switch (httpExchange.getRequestMethod()) {
        case "GET":
            handleGet(httpExchange);
            break;
        case "POST":
            handlePost(httpExchange);
            break;
        case "PUT":
            handlePut(httpExchange);
            break;
        case "DELETE":
            handleDelete(httpExchange);
            break;
    }
}

} else {
    // Return 405 Method Not Allowed for other request methods
    httpExchange.sendResponseHeaders(405, -1);
}
}

private void handleGet(HttpExchange httpExchange) throws IOException { //todo only keep handleGet
    URI uri = httpExchange.getRequestURI();
    byte[] responseBody = new byte[0]; //default body for malformed requests
    int responseCode = 404; //default status code for malformed requests

    if (uri.getPath().equals("/cords")) {
        //return all persons

        responseBody = Coordinates.listToJson(coordinatesService.get()).getBytes("UTF-8");
        responseCode = 200;
    } else if (uri.getPath().matches("/cords/[0-9]*")) { // todo not needed
        //return person with given id
        String[] split = uri.getPath().split("/");
        Coordinates coordinates = coordinatesService.get(Integer.parseInt(split[2]));
        if (coordinates != null) {
            responseBody = coordinates.toJson().getBytes("UTF-8");
            responseCode = 200;
        }
    }

    httpExchange.getResponseHeaders().set("content-type", "application/json");
    httpExchange.sendResponseHeaders(responseCode, responseBody.length);
    try (OutputStream os = httpExchange.getResponseBody()) { //write and close the response body
        os.write(responseBody);
    }
}

private void handlePost(HttpExchange httpExchange) throws IOException {

```

```
//return status code 501 - no implemented
httpExchange.sendResponseHeaders(501, -1);
}

private void handlePut(HttpExchange httpExchange) throws IOException {
    //return status code 501 - no implemented
    httpExchange.sendResponseHeaders(501, -1);
}

private void handleDelete(HttpExchange httpExchange) throws IOException {
    //return status code 501 - no implemented
    httpExchange.sendResponseHeaders(501, -1);
}
}
```

Programska potpora komunikacijskim sistavima

7. predavanje, 3. svibnja 2023.

doc. dr. sc. Jelena Božek

Tehnologije weba (HTTP, HTML, CSS, JS)



- Predavanja izradio izv. prof. dr. sc. Marin Vuković, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Creative Commons



- **slobodno smijete:**

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **remiksirati** — 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 licencem koja je ista ili slična ovoj.

U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela. Najbolji način da to učinite je linkom na ovu internetsku stranicu.

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 licencije preuzet je s <http://creativecommons.org/>.

Sadržaj predavanja

- Web
- Protokol HTTP
- HTML
- CSS
- JavaScript

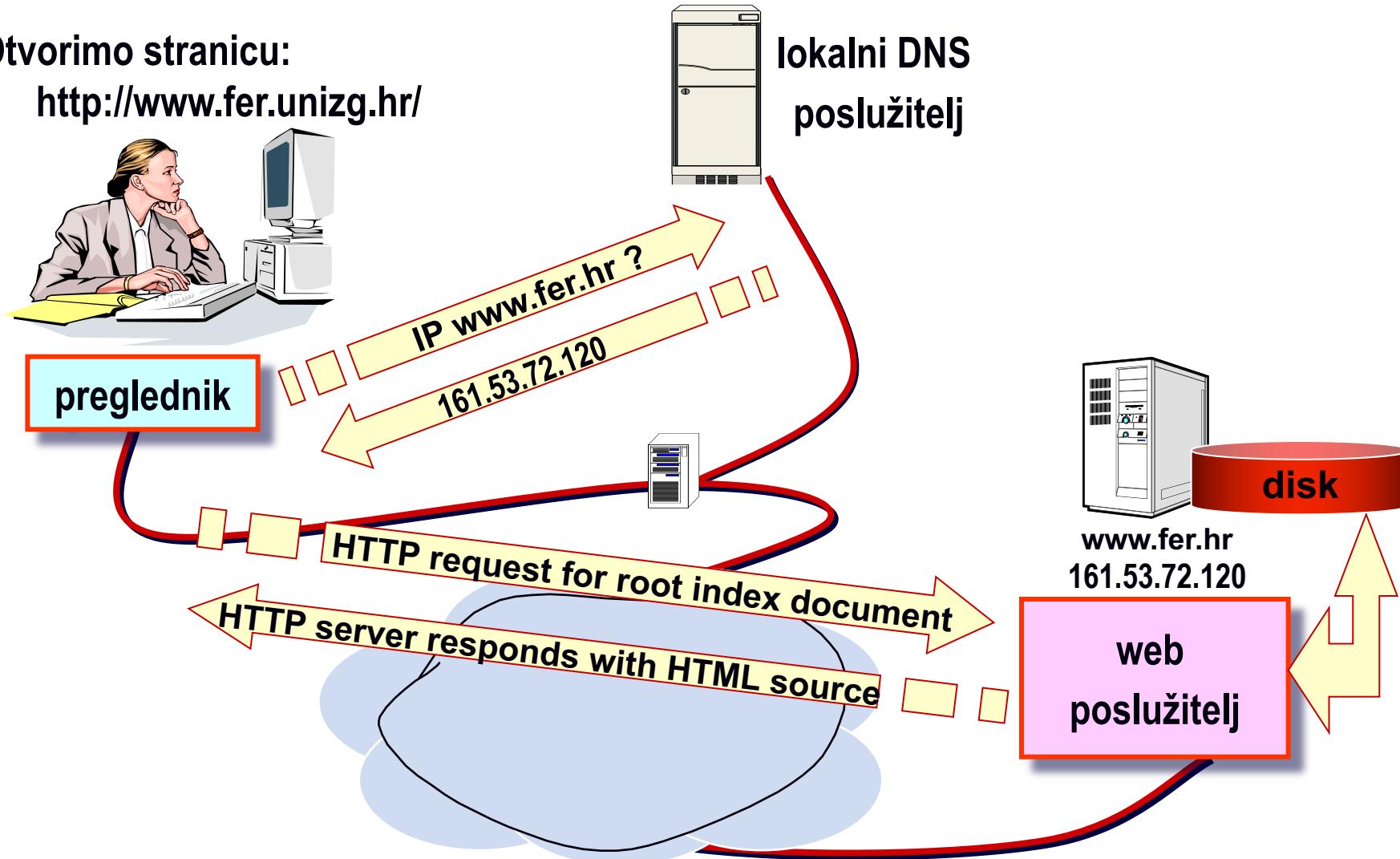
Web

- WWW – World Wide Web

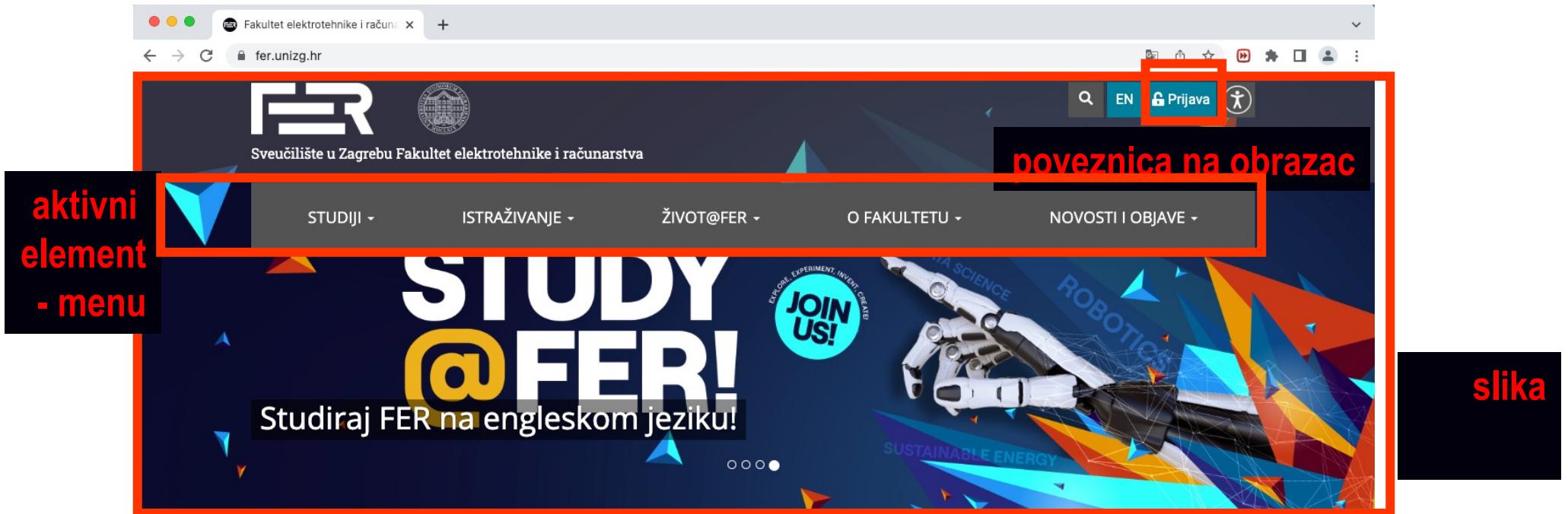
Kako radi WWW – osnovna ideja

Otvorimo stranicu:

<http://www.fer.unizg.hr/>



Primjer: Osnovna stranica FER-a – elementi



Studiji



PREDIPLOMSKI ST

tekst

Studij je organiziran kroz dva trogodišnja studijska programa (smjera), a započinje zajedničkom prvom godinom.



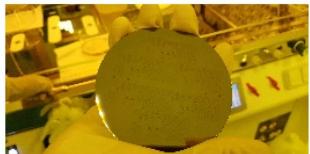
DIPLOMSKI STUDIJ

Diplomski studij organizira se kroz tri studijska programa, traje dvije godine, a izvode se po sustavu preduvjeta.



DOKTORSKI STUDIJ

Fakultet je nositelj doktorskog studija iz područja tehničkih znanosti, znanstvenog polja elektrotehnike i znanstvenog polja računarstva.



SPECIJALISTIČKI STUDIJI

Provodimo šest poslijediplomskih specijalističkih studija na kojima se stručnjake iz gospodarstva poučava novim tehnologijama.

(css - stil)

Procesiranje izvornog koda u HTML-u

- U ovom primjeru HTML sadrži:

- CSS - stil

```
<link rel="stylesheet" type="text/css" href="/_pub1683014340/themes_static/fer2016/default/style.css">
```

- slike

```

```

- JavaScript

```
<script type="text/javascript" src="/lib109/json2.js"></script>
```

- tekst

```
<p class="block_link_abstract">Studij je organiziran kroz dva ... </p>
```

- ostali elementi (hiperlinkovi, nabranja, obrasci, ...)

- Kada se procesira HTML potrebno je dohvatiti i druge elemente (slike, stil, JavaScript)

- Svi elementi se s poslužitelja dohvaćaju protokolom HTTP

- Klijent može pokrenuti novu konekciju ili koristiti postojeću za dohvaćanje ostalih elemenata

Protokol HTTP

- Hyper Text Transfer Protocol

Protokol Hypertext Transfer Protocol (HTTP)

- internetski protokol aplikacijskog sloja
- definira format i način razmjene poruka
 - tekstualan zapis, sličan formatu e-mail poruke i standarda MIME
- vrste poruka:
 - zahtjev ("metoda")
 - definira operaciju (metodu), resurs, protokol
 - naziv "metoda" potječe iz područja objektno-orientiranog programiranja
 - odgovor (rezultat)
 - rezultat (uspjeh, neuspjeh, greška,...) opisan statusnim kôdom
 - neke vrste odgovora u tijelu imaju sadržaj zatraženog resursa

Primjer HTTP-zahtjeva

```
1. GET /predmet/rassus HTTP/1.1  
2. Host: www.fer.unizg.hr  
3. User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:7.0.1) Gecko/20100101  
Firefox/7.0.1  
4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8  
5. Accept-Language: hr,en-us;q=0.7,en;q=0.3  
6. Accept-Encoding: gzip, deflate  
7. Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7  
8. Connection: keep-alive  
9. Cache-Control: max-age=0
```

prazan redak

**tijelo
poruke**

Primjer HTTP-zahtjeva

```
1. GET /predmet/rassus HTTP/1.1
2. Host: www.fer.unizg.hr
3. User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:7.0.1) Gecko/20100101
   Firefox/7.0.1
4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5. Accept-Language: hr,en;q=0.7,en-us;q=0.3
6. Accept-Encoding: gzip, deflate
7. Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
8. Connection: keep-alive
9. Cache-Control: max-age=0
```

Obavezni dijelovi

Metode zahtjeva

- metoda zahtjeva određuje što se traži od resursa
- HTTP/1.1 definira 8 metoda i omogućuje dodavanje novih metoda (extensions):
 - OPTIONS - informiranje i mogućnostima resursa i poslužitelja
 - GET - za dohvaćanje resursa (najčešća)
 - HEAD - za dohvaćanje podataka o resursu (npr. veličina, postojanje)
 - POST - aktiviranje resursa (npr. slanje podataka obrazaca)
 - PUT - postavljanje entiteta (npr. promjena podataka - kod REST-a)
 - DELETE - brisanje resursa (npr. kod REST-a)
 - TRACE - za dijagnostiku
 - CONNECT - za buduću uporabu (ne implementira se)

Primjer HTTP-odgovora

```
1. HTTP/1.1 200 OK  
2. Date: Wed, 12 Oct 2011 08:19:32 GMT  
3. Server: Apache/2.2.20 (FreeBSD) mod_ssl/2.2.20 OpenSSL/0.9.8q mod_fcgid/2.3.6  
4. Expires: Thu, 19 Nov 1981 08:52:00 GMT  
5. Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0  
6. Pragma: no-cache  
7. P3P: CP="NOI CURa ADMa DEVa TAI PSAa PSDa TVAa IVDa HISa OTPa OUR BUS IND UNI COM NAV INT"  
8. Set-Cookie: CMS=2p72ge55hqm3; expires=Wed, 19-Oct-2011 20:19:32 GMT; path=/;  
domain=www.fer.unizg.hr; HttpOnly  
9. Vary: Accept-Encoding  
10. Transfer-Encoding: chunked  
11. Content-Type: text/html; charset=utf-8  
12.  
13. d9e7  
14. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
15. <html xmlns="http://www.w3.org/1999/xhtml">  
16. prazan redak  
17. <head>  
18. <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
19. ...  
tijelo poruke
```

HTTP-odgovor

- početni redak (protokol, statusni kod i opis)
- kategorije statusnih kodova:
 - **1xx** – Informativne - ni uspjeh, ni neuspjeh
 - **2xx** – Uspjeh - poslužitelj je primio, razumio i ispunio zahtjev
 - **3xx** – Preusmjeravanje - potrebno poduzeti dodatne akcije
 - **4xx** – Greška na klijentu - zahtjev je neispravan
 - **5xx** – Greška na poslužitelju - zahtjev je ispravan, ali ga poslužitelj ne može ispuniti
- u tijelu odgovora se obično prenosi reprezentacija resursa (“entitet”) koju preglednik treba prikazati korisniku (npr. HTML)
- primjer nekih polja zaglavlja:
 - **Content-Type**: format entiteta
 - **Content-Length**: duljina entiteta u tijelu (u oktetima)

HTML

- Hyper Text Markup Language

World Wide Web Consortium (W3C)

- www.w3.org
- Aktivnosti organizirane u [radne grupe](#)
- HTML i CSS
 - <https://www.w3.org/standards/webdesign/htmlcss>
- Edukacija: <http://www.w3schools.com>

Povijest HTML-a

- 1991. nacrt u CERN-u
- 1993. javna pojava HTML-a
- 1995. HTML 2
 - RFC 1866 + dodatni RFC-ovi
- 1996. HTML 3
 - IETF odustaje od standardizacije HTML-a
- 1997. HTML 3.2
 - izdaje ga W3C
- 1998. HTML 4
 - Podrška za video sadržaj (Flash)
- 2000. XHTML 1.0 ekvivalent HTML 4.01 Strict
- 2004. počinje standardizacija HTML 5
 - 2008. prvi javni nacrt
 - 2012. radni nacrt
 - 28.10.2014. postao standard
- 2014.-sada – radi se na poboljšanju
 - 5.2. standard 14.12.2017.
 - Living Standard – kontinuirano

Primjer HTML-dokumenta

- <!DOCTYPE html> deklaracija
- <html> korijenski element
- <head> zaglavlje s metapodacima
- <title>Naslov</title> naslov
- </head>
- <body> tijelo – vidljivi elementi
- <h1>Naslov</h1> veliki naslov
- <p>Neki tekst.</p> odломак teksta
- </body>
- </html>

Oznake

<oznaka> - otvarajuća

</oznaka> - zatvarajuća

<oznaka/> - otvarajuća i zatvarajuća

- U HTML-u možemo imati i samo otvarajuću oznaku bez zatvarajuće
 - Npr.

- Atributi:
 - Npr. ...

Važni atributi

- **id** – jedinstveni identifikator oznake
- **name** – ime oznake, ali ne treba biti jedinstveno
- **class** – jedna ili više klase oznake (referira se na klasu u stilu)
- **style** – opisuje CSS stil u HTML dokumentu

Popis bitnih oznaka

- h1, h2, h3, ... h6 – naslovi
- p – odlomak
- Oznake za formatiranje:
 - b – podebljano
 - strong – važni tekst
 - i – koso
 - em – naglašeno
- Komentar <!-- -->
- Poveznice: tekst
- Slika:
 -
- div, span – služe za označavanje dijela dokumenta
- br, hr – prelazak u novi red i horizontalna linija

Tablica

```
<table style="width:100%">  
  <tr>  
    <th>Ime</th>  
    <th>Prezime</th>  
    <th>Broj telefona</th>  
  </tr>  
  <tr>  
    <td>Pero</td>  
    <td>Perić</td>  
    <td>203</td>  
  </tr>  
  ...  
</table>
```

The diagram illustrates the structure of an HTML table with the following annotations:

- A blue arrow points from the word "tablica" to the opening tag "<table style="width:100%">".
- A blue arrow points from the word "red" to the opening tag "<tr>".
- A blue arrow points from the word "element zaglavlja" to the first row of column headers: "<th>Ime</th>", "<th>Prezime</th>", and "<th>Broj telefona</th>".
- A blue arrow points from the word "element u redu" to the first row of data cells: "<td>Pero</td>", "<td>Perić</td>", and "<td>203</td>".

Liste

Neporedana lista

```
<ul>
  <li>Meso</li>
  <li>Luk</li>
  <li>Salata</li>
</ul>
```

- Meso
- Luk
- Salata

Poredana lista

```
<ol>
  <li>Meso</li>
  <li>Luk</li>
  <li>Salata</li>
</ol>
```

1. Meso
2. Luk
3. Salata

Opisna lista

```
<dl>
  <dt>Meso</dt>
  <dd>- teletina</dd>
  <dt>Luk</dt>
  <dd>- crveni</dd>
</dl>
```

Meso

- teletina

Luk

- crveni

CSS

- Cascading Style Sheets

CSS

- Opisuje kako će biti prikazani elementi iz HTML-a na ekranu, papiru ili nekom drugom mediju
- Sintaksa:

```
selektor {  
    svojstvo1:vrijednost1;  
    svojstvo2:vrijednost2;  
    ...  
}
```

Selektori

- #id – odabire oznaku s identifikatorom id
- .klasa – odabire oznaku po klasi
- oznaka – odabire elemente po oznaci
- Npr. p.klasa – odabire oznake p koje imaju atribut klasa
- Grupiranje – odabire neki od selektora
 - Npr. h1, h2 – odabire oznake h1 i h2
 - Npr. h1 h2 – odabire h2 koji je unutar h1

Uključivanje stilova u dokument

- Eksterni stilovi
 - <head>
 - <link rel="stylesheet" type="text/css" href="neki.css">
 - </head>
- Interni stilovi
 - <head>
 - <style>
 - ...
 - </style>
 - </head>
- Stilovi u oznaci
 - <h1 style="color:blue">Naslov</h1>

Primjer (HTML): primjer.html

```
<!DOCTYPE html>
<html>
<head>
<title>Primjer</title>
<link rel="stylesheet" href="primjer.css">
</head>
<body>
<h1>Tekst u boji</h1>
<p>Sljedeći elementi imaju tekst u boji:</p>

<div class="text-red">
<h2>Crveno</h2>
<p>Tekst u crvenoj boji.</p>
</div>
<div class="text-blue">
<h2>Plavo</h2>
<p>Ovaj tekst je u plavoj boji.</p>
</div>
</body>
</html>
```

Primjer (CSS): primjer.css

```
.text-red {  
    color:#f44336  
}  
  
.text-blue {  
    color:#2196F3  
}
```

Tekst u boji

Sljedeći elementi imaju tekst u boji:

Crveno

Tekst u crvenoj boji.

Plavo

Ovaj tekst je u plavoj boji.

JavaScript

Skripte na klijentu

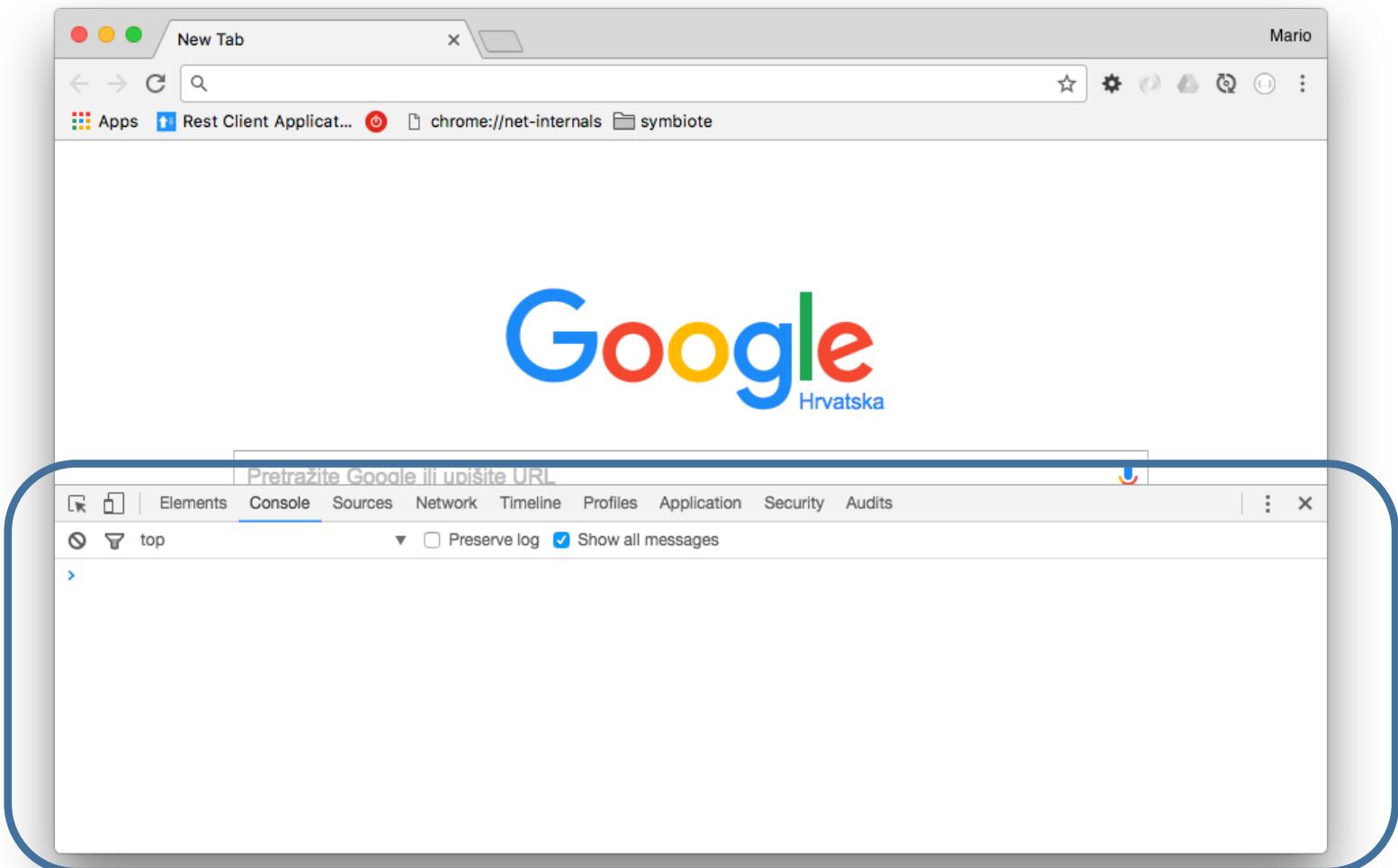
- uključene u HTML ili u posebnoj datoteci koja je povezana
- obično se koristi:
 - JavaScript (Netscape), JScript (Microsoft), ECMAScript
- ECMAScript – standardiziran
 - specifikacije ECMA-262 i ISO/IEC 16262
 - svojstva: dinamički, slabo povezan, objektni, funkcijski
- nema veze s Javom
- svrha:
 - dinamički elementi
 - interakcija s korisnikom
 - provjera obrazaca
 - komunikacija s poslužiteljem (AJAX)
 - ...
- tutorial: <http://www.w3schools.com/JS/>

JavaScript (ECMAScript)

- Poboljšanja u preglednicima
 - Alati za debuggiranje
 - `console.log()` – ispis u posebnu konzolu
 - Objekt `window.JSON`
 - Metoda `parse` – korište je ugrađenog parsera koji je puno brži
 - Metoda `stringify` – pretvaranje objekta u string
 - Jedinstveno rukovanje događajima u svim preglednicima
 - `addEventListener()` – dodavanje funkcije za obradu događaja
 - `dispatchEvent()` – kreiranje novog događaja
- Verzije ([podrška](#)):
 - ECMAScript 5 – podržavaju je većina preglednika
 - ECMAScript 6 – 2015. – to bi obično željeli koristiti
 - ECMAScript 7 – 2016.
 - ECMAScript 8 – 2017.
 - ECMAScript 9 – 2018.
 - ECMAScript 10 – 2019.

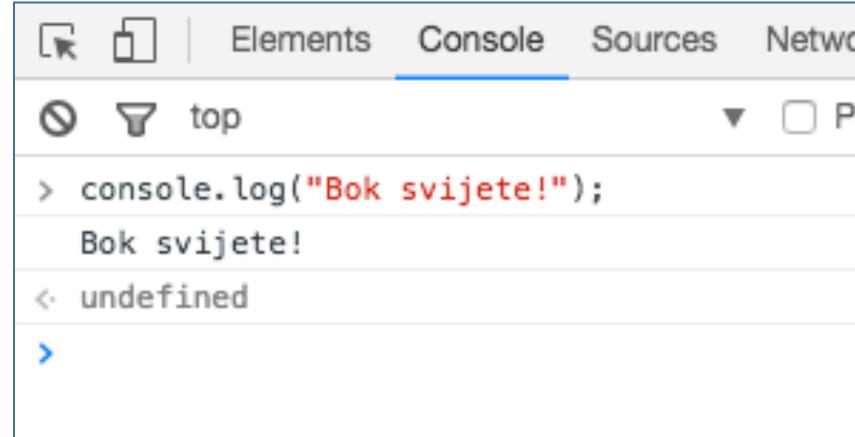
JavaScript u Chromeu

- View → Developer → JavaScript Console



Rad u konzoli preglednika

- console.log("Bok svijete!");

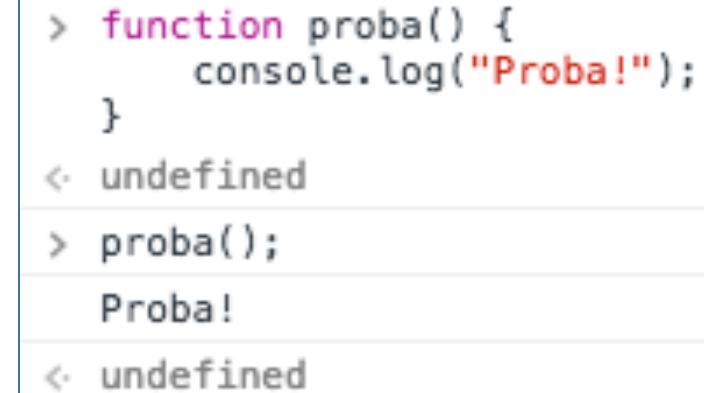


```
> console.log("Bok svijete!");
Bok svijete!
<- undefined
>
```

// deklariranje globalne funkcije

```
function proba() {
    console.log("Proba!");
}

// poziv funkcije
proba();
```



```
> function proba() {
    console.log("Proba!");
}
<- undefined
> proba();
Proba!
<- undefined
```

Varijable

```
var a = 5; // deklariraju se pomoću ključne riječi var
```

```
var b = a * 2;
```

```
console.log(b);
```

```
b = a * 3;
```

```
console.log(b);
```

// funkcije mogu biti spremljene u varijable

```
var f = function(x) {
```

```
    return x + 10;
```

```
}
```

```
console.log(f(3));
```

Tipovi podataka

```
var n = 5; // brojevi
```

```
n = 5.6;
```

```
var s = "Neki tekst!"; // string
```

```
var o = {firstName:"Ivan", lastName:"Horvat"}; // objekt
```

```
o.firstName;
```

```
var b = true; // boolean
```

```
b = false;
```

```
// polje koje je zapravo objekt
```

```
var a = [1, "Auto", {owner: o}];
```

```
a[0];
```

```
a.length;
```

```
// operator typeof
```

```
typeof a;
```

JavaScript u HTML-u

- Dodamo oznaku script u head ili body

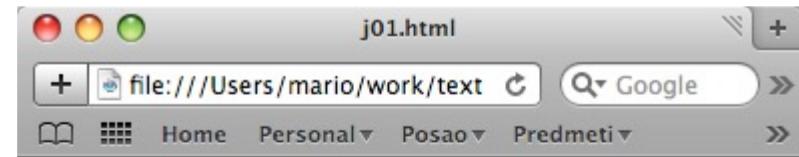
```
<html>  
<body>  
  
<script type="text/javascript">  
document.write("Ovo je napisano iz prvog JavaScripta!");  
</script>
```

```
</body>  
</html>
```

- Uključivanje koda iz datoteke

```
<script type="text/javascript" src="skripta.js"></script>
```

- U HTML-u 5 se type može izostaviti (podrazumijevana vrijednost)



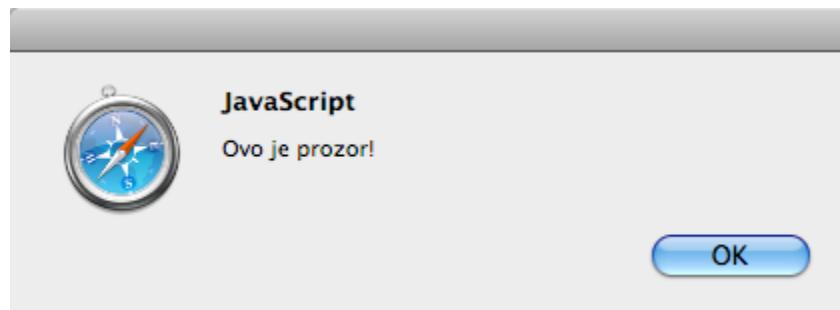
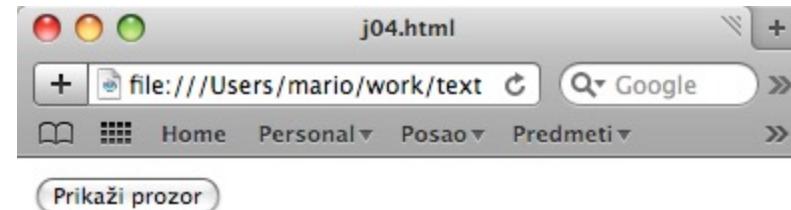
Ovo je napisano iz prvog JavaScripta!

Prozori

```
<html>
<head>
<meta charset="UTF-8" />
<script>
function disp_alert() {
    alert("Ovo je prozor!");
}
</script>
</head>
<body>

<input type="button" onclick="disp_alert()"
       value="Prikaži prozor" />

</body>
</html>
```

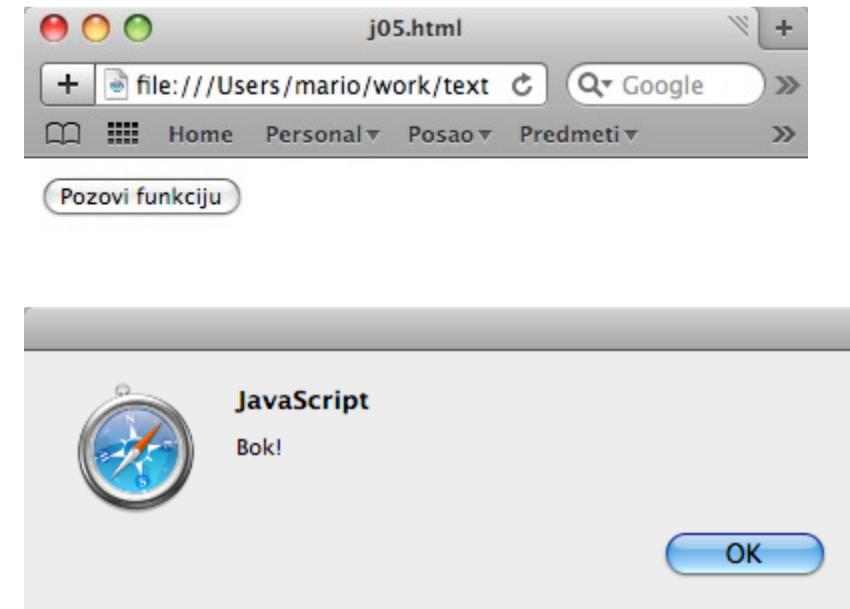


```
confirm("pitanje"); // vraća true/false
// vraća upisani odgovor
prompt("pitanje", "ponuđeni odgovor");
```

Funkcije s parametrima i povezivanje s gumbom

```
<html>
<head>
<meta charset="UTF-8" />
<script>
function myfunction(txt) {
    alert(txt);
}
</script></head>
<body>

<form>
<input type="button"
onclick="myfunction('Bok!')" value="Pozovi funkciju">
</form>
</body>
</html>
```



Pojednostavljen API za odabir elemenata u HTML-u 5

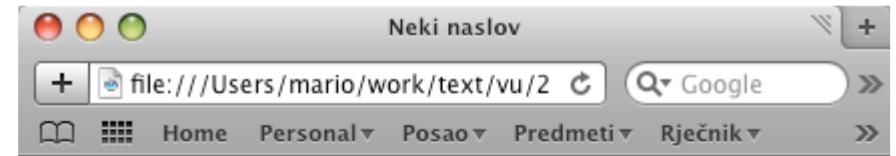
- `getElementById()` – atribut `id`
 - `getElementsByName()` – atribut `name`
 - `getElementsByTagName()` – ime elementa
-
- `querySelector()` – prvi element s određenim CSS-om
 - `querySelectorAll()` – svi elementi s određenim CSS-om

Upravljanje elementima

```
<html><head><title>Neki naslov</title></head>
<body><script>
document.write("Naslov:" + document.title + "<br>");

function addDiv() {
    x = document.getElementById("mojId");
    x.innerHTML = x.innerHTML + "*";
}

function addSpan() {
    x = document.getElementsByName("imeOznake");
    x[0].innerHTML = x[0].innerHTML + "$";
}
</script>
<div id="mojId" onclick="addDiv()">div: </div><br>
<span name="imeOznake" onclick="addSpan()">span: </span>
</body>
</html>
```



Naslov:Neki naslov
div: *
span: \$\$\$

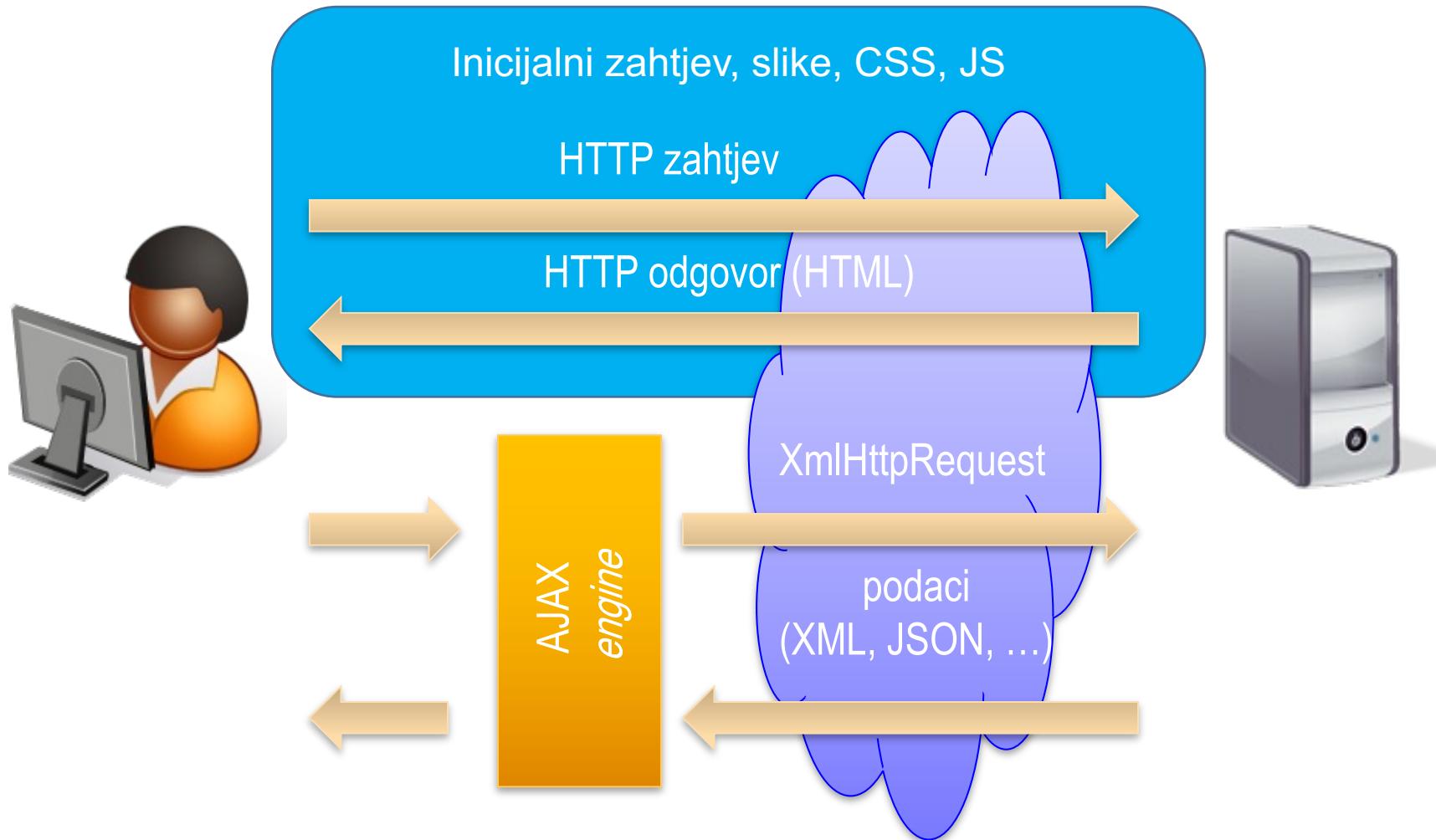
Javascript u HTML-u 5

- Poboljšanja u preglednicima
 - Alati za debuggiranje
 - `console.log()` – ispis u posebnu konzolu
 - Objekt `window.JSON`
 - Metoda `parse` – korištenje ugrađenog parsera koji je puno brži
 - Metoda `stringify` – pretvaranje objekta u string
 - Jedinstveno baratanje događajima u svim preglednicima
 - `addEventListener()` – dodavanje funkcije za obradu događaja
 - `dispatchEvent()` – kreiranje novog događaja

Poznate knjižnice u JavaScriptu

- ◆ popis se stalno mijenja i stalno raste
 - <https://www.javascripting.com>
- ◆ DOM manipulation -
<https://www.javascripting.com/dom/>
- ◆ GUI - radni okvir -
<https://www.javascripting.com/user-interface/>

AJAX (Asynchronous JavaScript and XML)



Programska potpora komunikacijskim sustavima

8. predavanje, 10. svibnja 2023.

• doc. dr. sc. Jelena Božek

Arhitekturni stil REST



- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Općenito o REST-u
- Ograničenja koja uvodi REST
- Resursi u REST-u
- CRUD resursa u REST-u korištenjem metoda HTTP zahtjeva
- Primjer REST-usluge

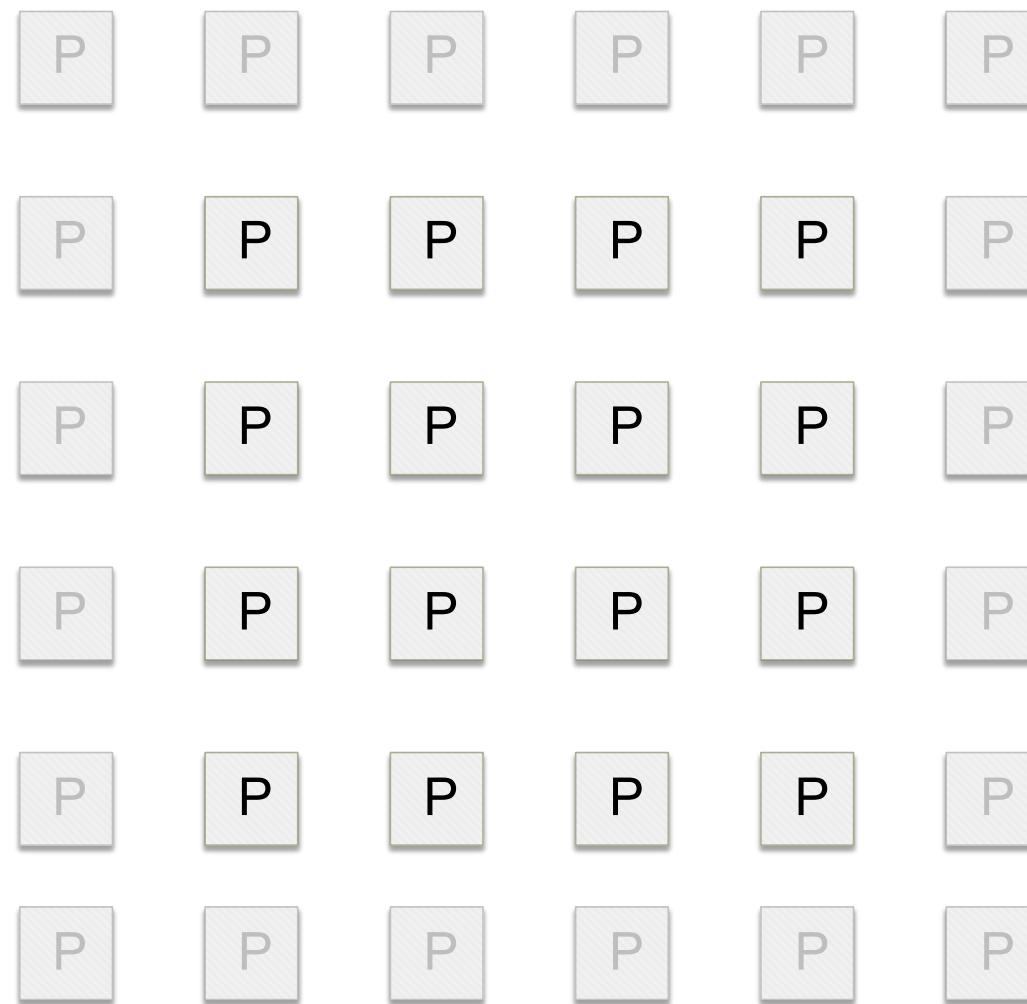
REST (*Representational State Transfer*)

- Hrvatski prijevod: **prijenos prikaza stanja resursa**
- Pojam je 2000. godine skovao [Roy Fielding](#) u svojoj doktorskoj disertaciji „[Architectural Styles and the Design of Network-based Software Architectures](#)“
- Nije standard već je **arhitekturni stil** za *World Wide Web* (WWW)
 - Neovisan je o konkretnoj implementaciji tj. tehnologiji
 - Uvodi ograničenja kojih se trebamo držati pri razvoju i implementaciji programskih komponenti na Webu
 - Definira način na koji trebaju komunicirati raspodijeljeni procesi na Webu
- RESTful API (*application programming interface*) je API koji poštuje ograničenja arhitekturnog stila REST i omogućava interakciju s RESTful web-uslugama

Web-usluga ≠ web-aplikacija

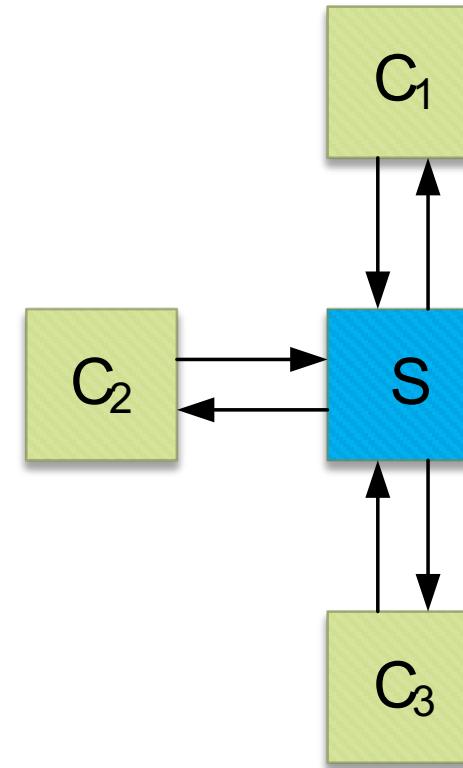
- Web-aplikacija
 - Tema prošlotjednog predavanja
 - Aplikacija kojoj se pristupa putem web-preglednika koji radi na računalu krajnjeg korisnika
 - Namijenjena je ljudima
- Web-usluga
 - Tema današnjeg predavanja
 - Usluga koja omogućuje međusobnu interakciju **raspodijeljenih procesa** na Webu
 - Namijenjena je (procesima na) računalima

Raspodijeljeni procesi na Webu



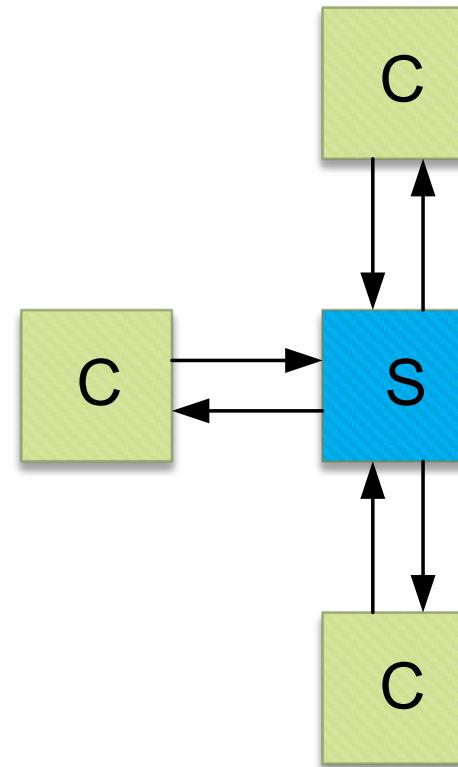
Ograničenje 1: model klijent-server

- Jasna podjela odgovornosti (*separation of concerns, SoC*) na klijenta i poslužitelja
 - Ne rade svi sve
 - Odgovornost za korisničko sučelje je razdvojeno od odgovornosti za pohranu podataka
- Prednosti ovog ograničenja
 - Pojednostavljen je razvoj programskih komponenti
 - Poboljšava se skalabilnost usluge
 - Olakšane su programske nadogradnje



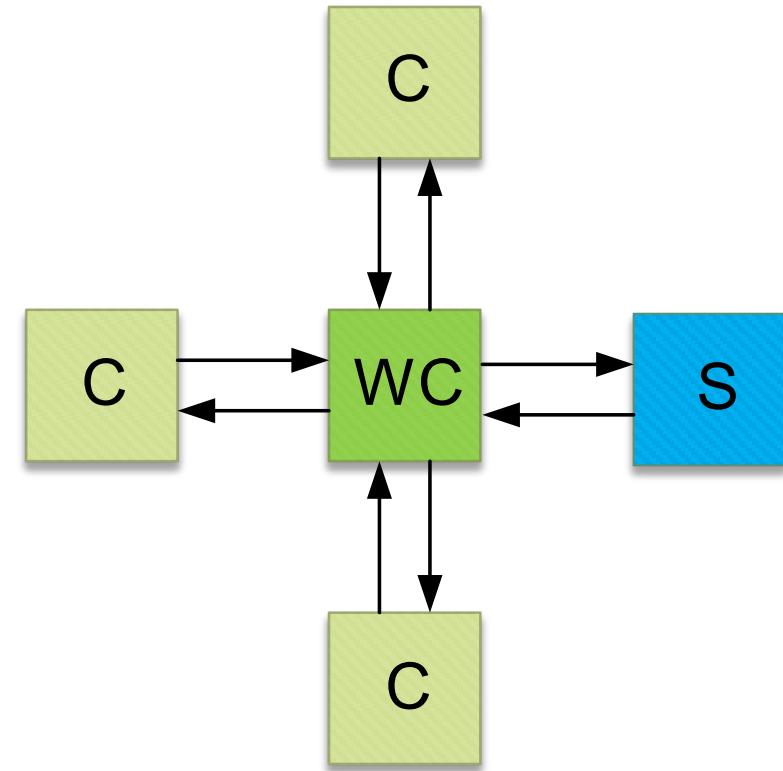
Ograničenje 2: Nema stanja

- Poslužitelj ne pamti stanje komunikacije s klijentom
 - Poslužitelj ne pravi razliku između klijenata i njihovih pojedinačnih zahtjeva
 - Svaki zahtjev mu je neovisan o prethodnim zahtjevima
 - Zahtjev u sebi sadrži sve informacije koje su potrebne poslužitelju da napravi odgovor
- Prednosti ovog ograničenja
 - Poslužitelj je rasterećen jer ne mora pamtiti stanje komunikacije (*session*) sa svakim pojedinim klijentom
 - Poboljšane su performanse i skalabilnost poslužiteljske komponente



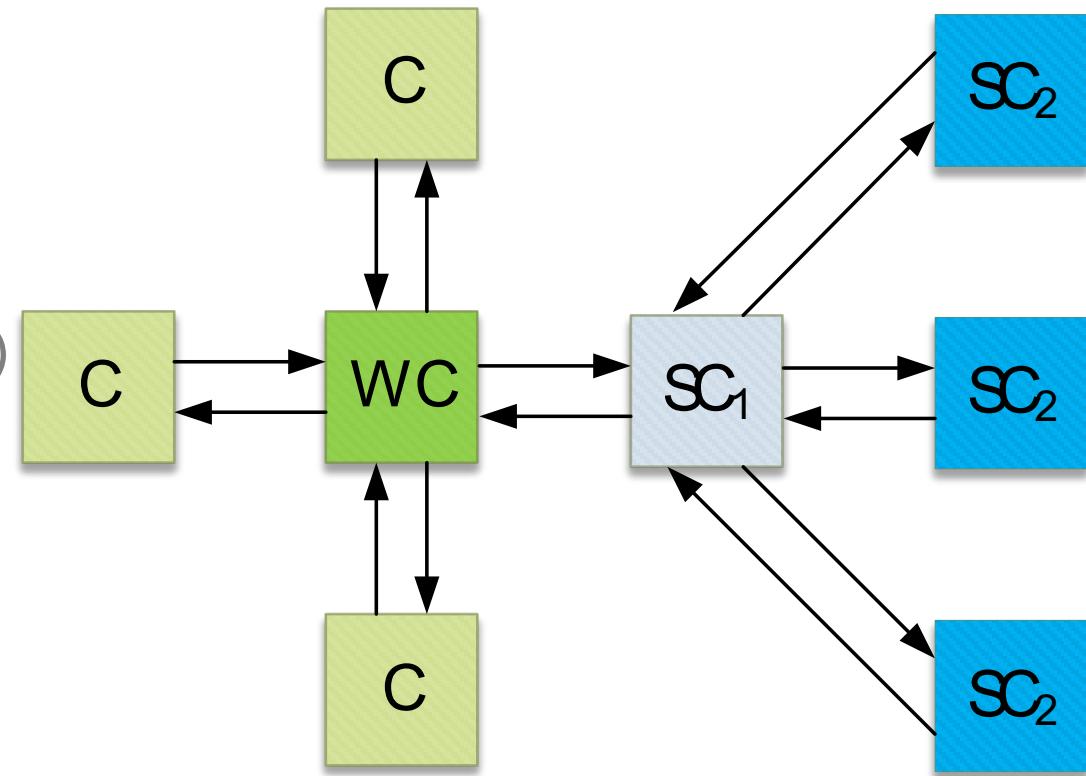
Ograničenje 3: Pamtljivi odgovori

- Odgovori moraju moći biti pamtljivi (*cacheability*) tj. moraju se moći pohraniti u priručno spremište
 - Privremena spremišta se mogu nalaziti kod klijenata ili kod posebnih spremišta na Webu (*Web Cache, WC*) koja su smještena između klijenata i poslužitelja
 - Svaki odgovor mora sadržavati informaciju o tome smije li se privremeno pohraniti ili ne
- Prednosti ovog ograničenja
 - Poslužitelj je rasterećen jer ne mora odgovarati na sve zahtjeve
 - Poboljšane su performanse i skalabilnost poslužiteljske komponente



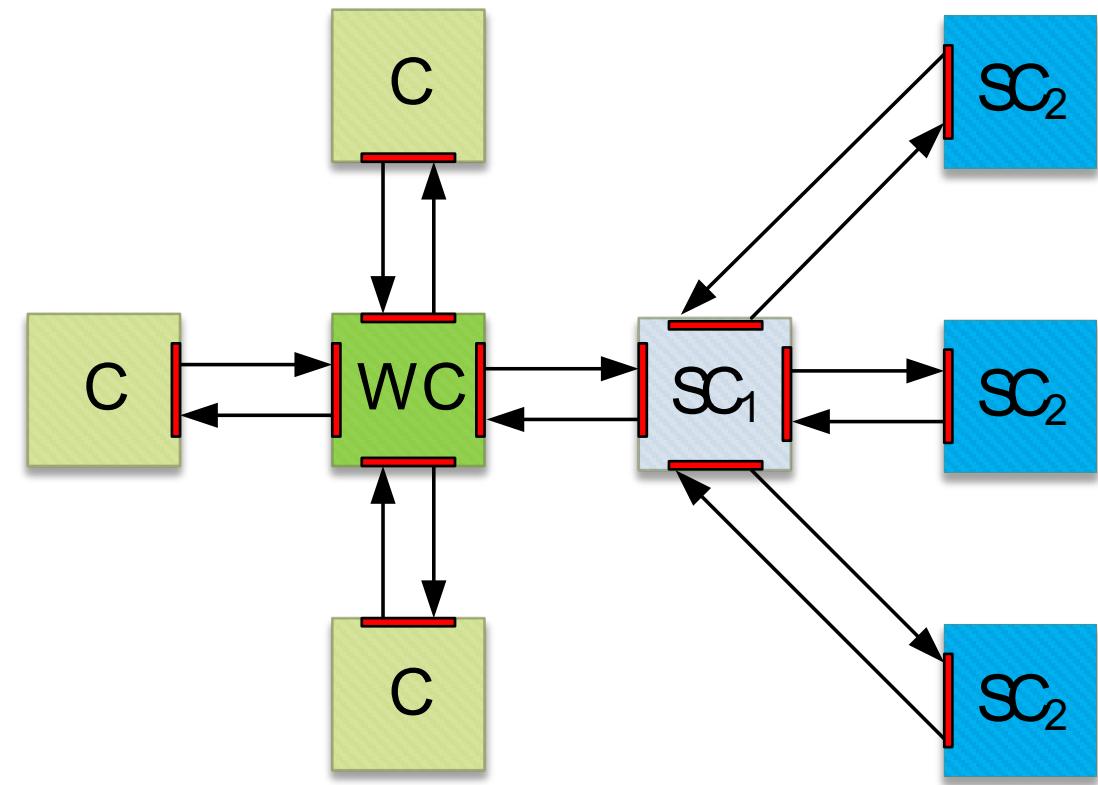
Ograničenje 4: Slojevitost poslužitelja

- Arhitektura poslužiteljske komponente može biti slojevita
 - Između poslužitelja i klijenta se moraju moći nalaziti poslužiteljske komponente kao što su zastupnik (*proxy*) ili uravnoteživač opterećenja (*load balancer*)
 - Slojevitost arhitekture poslužitelja ne smije zahtijevati izmjene programskog koda poslužitelja i klijenata
- Prednosti ovog ograničenja
 - Uravnoteživanjem opterećenja su poboljšane performanse i skalabilnost poslužiteljskog dijela



Ograničenje 5: Uniformo sučelje

- Sučelje između komponenti treba biti uniformno
 - Identifikacija resursa
 - Manipulacija resursa njihovom reprezentacijom
 - Samoopisujuće poruke
 - Princip HATEOAS
- Prednosti ovog ograničenja
 - Olakšan razvoj programskih komponenti



Identifikacija resursa

- U praksi se koristi URI (*Uniform Resource Identifier*)
- Sintaksa URI-ja:

```
scheme ":" [://" authority] path ["?" query] ["#" fragment]
```

```
authority = [userinfo "@" host [":" port]]
```

- Primjeri:
 - <http://xyz.com/users/> - popis svih korisnika
 - <http://xyz.com/users/1> - korisnik s identifikatorom 1
 - <http://xyz.com/users/1/name> - ime korisnika s identifikatorom 1

Reprezentacija resursa

- Pojedini resurs može imati više reprezentacija (xml, html, json, jpg)

application/xml

```
<user>
  <id>1</id>
  <name>Ivo</name>
  <surname>Ivić</surname>
</user>
```

application/json

```
{
  "id":1,
  "name":"Ivo",
  "surname":"Ivić"
}
```

text/html

```
<p class="user">
  <span class="id">1</span>
  <span class="name">Ivo</span>
  <span class="surname">Ivić</span>
</p>
```

image/jpeg



Samoopisujuće poruke

- Svaka poruka koja se razmjenjuje opisuje način na koji je treba parsirati tj. pročitati i protumačiti
- U praksi se koristi tip medija (*media type*) u standardnom obliku kako ga klasificira organizacija [Internet Assigned Numbers Authority \(IANA\)](#)
- Sintaksa tipa medija:
type "/" [tree "."] subtype ["+" suffix]* [";" parameter]
- Primjeri tipa medija
 - application/json
 - application/xml
 - application/html
 - image/jpeg

Princip HATEOAS

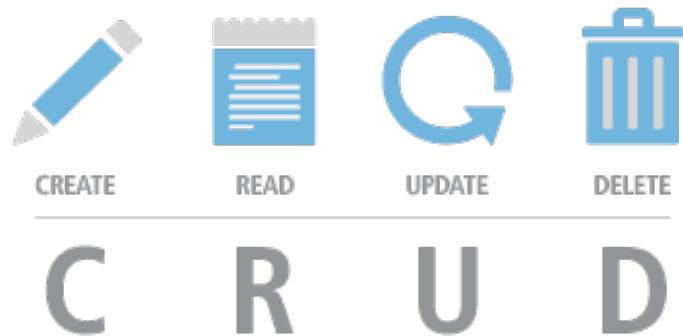
- HATEOAS (*Hypermedia As The Engine Of Application State*)
- Osnovno znanje o hipermediji je dovoljno za interakciju s poslužiteljem
- Aplikacijski poslužitelj klijentu daje informacije koristeći hipermediju
- Hipermedija je nelinearni medij informacije koji uključuje grafiku, audio, video, tekst i poveznice (*hyperlinks*)

HTTP/1.1 200 OK

```
{  
  "account": {  
    "account_number": 12345,  
    "balance": {  
      "currency": "usd",  
      "value": 100.00  
    },  
    "links": {  
      "deposits":  
        "/accounts/12345/deposits",  
      "withdrawals":  
        "/accounts/12345/withdrawals",  
      "transfers":  
        "/accounts/12345/transfers",  
      "close-requests":  
        "/accounts/12345/close-requests"  
    }  
  }  
}
```

CRUD

- REST u praksi koristi metode HTTP zahtjeva da bi radio CRUD operacije nad resursima



HTTP request method	CRUD
GET	READ
<i>HEAD</i>	
POST	CREATE
PUT	(FULL) UPDATE
DELETE	DELETE
<i>CONNECT</i>	
<i>OPTIONS</i>	
<i>TRACE</i>	
PATCH	(PARTIAL) UPDATE

Svojstva metoda HTTP zahtjeva

- Sigurna (*safe*) - bez posljedica za podatke
- Indempotentna (*idempotent*) - može se izvršavati više puta
- Pamtljiv odgovor (*cachable*)

No (RESTful API)

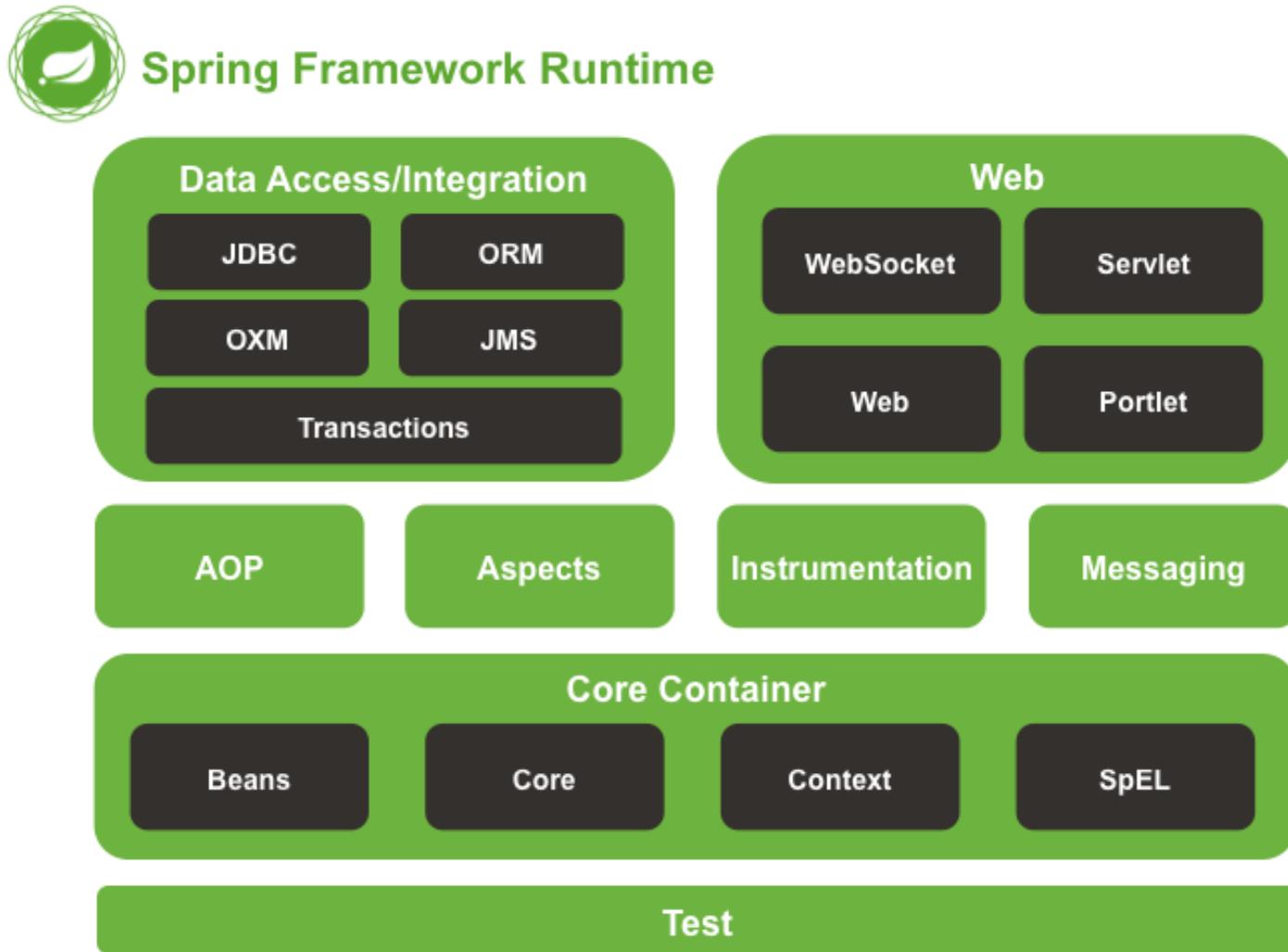
HTTP request method	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
GET	Optional	Yes	Yes	Yes	Yes
HEAD	Optional	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	Optional	Yes	No	Yes	No
CONNECT	Optional	Yes	No	No	No
OPTIONS	Optional	Yes	Yes	Yes	No
TRACE	No	Yes	Yes	Yes	No
PATCH	Yes	Yes	No	No	No



Spring - <http://spring.io>

- Radni okvir za lakši razvoj web-aplikacija
- Bavi se konfiguracijom objekata u sustavu
- Upravlja poslovnim objektima kao običnim objektima (*POJO – plain old java objects*)
- Brine se za kreiranje objekata (*IoC – inversion of control*)
- Povezuje kreirane objekte (*IoC, wiring up, 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

Springova arhitektura - dokumentacija



Spring Boot

- Jedan od Springovih podprojekata
 - <http://projects.spring.io/spring-boot/>
- Pojednostavljuje korištenje Springa
- 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

Primjer: Persons

- Popis nastavnika (osoba)
- Dohvaćanje i spremanje u memorija poslužitelja
- Prikaz u JSON-u (REST)
 - uređivanje (dodavanje, promjena, brisanje) podataka

Stvaranje Spring Boot projekta



- Otvoriti stranicu <https://start.spring.io>
- Odabratи:
 - Project: **Maven** Project (u primjeru u repozitoriju je Gradle Project)
 - Language: Java
 - Spring Boot: 2.6.7 (ili novije 3.0.6)
- Popuniti **Project Metadata**:
 - Group: hr.fer.zkist.ppks
 - Artifact: persons
 - Name: persons
 - Package name: hr.fer.zkist.ppks.persons
- Odabratи:
 - Packaging: Jar
 - Java Version: 17
- Odabratи **Dependencies**:
 - Spring Web
 - REST repositories
 - Spring HATEOAS
- Kliknuti na **Generate Project**
- Uvesti projekt u razvojnu okolinu (Apache NetBeans, Eclipse, IntelliJ IDEA)

Spring Boot projekt

- struktura projekta
- pogledati:
 - klasa PersonsApplication
 - klasa koja se pokreće
 - datoteka pom.xml
 - skripta za izgradnju

The screenshot shows an IDE interface with the following components:

- Projects View:** Shows a project named "persons" with a single source package "hr.fer.zkist.ppkbs.persons" containing the file "PersonsApplication.java". Other sections include Test Packages, Other Sources, Dependencies, Test Dependencies, and Project Files.
- Code Editor:** Displays the content of "PersonsApplication.java":

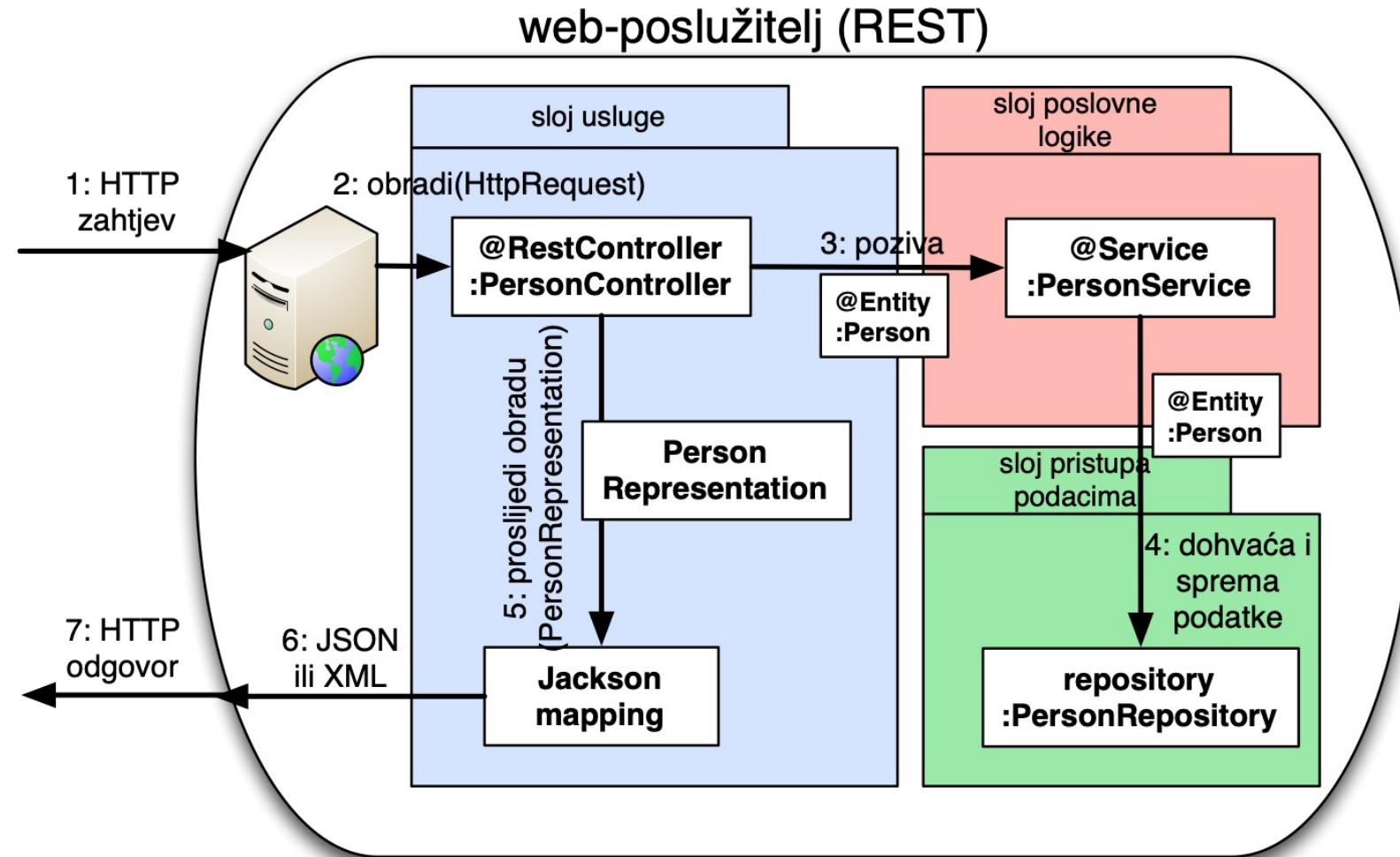
```
1 package hr.fer.zkist.ppkbs.persons;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class PersonsApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(PersonsApplication.class, args);
11     }
12 }
13
14 hr.fer.zkist.ppkbs.persons.PersonsApplication
```

- Navigator View:** Shows the members of the "PersonsApplication" class, including the constructor "PersonsApplication()" and the main method "main(String[] args)".
- Output View:** Shows the message "Unpacking index for Central Repository".

Primjer RESTful web-usluge

resurs	podržane metode	šalje	svrha
/persons	GET	osoba	vraća listu osoba
	POST		stvara novu osobu
/persons/{id}	GET	osoba	vraća osobu s ID-om
	DELETE		briše osobu
	PUT	osoba	mijenja podatke o osobi

Arhitektura RESTful web-usluge



Primjer upita i odgovora [1]

- GET /persons
- odgovor: 200 OK

```
[  
 {  
   "id": 0,  
   "firstName": "Goran",  
   "lastName": "Delač",  
   "phone": "01/6129-549",  
   "room": "D-339-2"  
 },  
 {  
   "id": 1,  
   "firstName": "Marin",  
   "lastName": "Šilić",  
   "phone": "01/6129-549",  
   "room": "D-339-2"  
 }  
 ]
```

Primjer upita i odgovora [2]

- POST /persons

- sadržaj zahtjeva:

```
{  
    "id": 4,  
    "firstName": "Mislav",  
    "lastName": "Grgić",  
    "phone": "01/6129-851",  
    "room": "C11-06"  
}
```

- odgovor: 201 Created

- header: "location" : "<http://localhost:8080/persons/4>"

Primjer upita i odgovora [3]

- GET /persons/4
- odgovor: 200 OK

```
{  
    "id": 4,  
    "firstName": "Mislav",  
    "lastName": "Grgić",  
    "phone": "01/6129-851",  
    "room": "C11-06"  
}
```

Primjer upita i odgovora [4]

- GET /persons/5
- odgovor: 404 Not Found

```
{  
  "timestamp": "2022-05-10T13:48:15.450+00:00",  
  "status": 404,  
  "error": "Not Found",  
  "path": "/persons/4"  
}
```

Primjer upita i odgovora [5]

- PUT /persons/4

```
{  
    "id": 4,  
    "firstName": "Mislav",  
    "lastName": "Grgić",  
    "phone": "01/6129-999",  
    "room": "C11-06"  
}
```

- odgovor: 204 No Content

Primjer upita i odgovora [6]

- PUT /api/persons/100

```
{  
  "id": 100,  
  "firstName": "Mislav",  
  "lastName": "Grgić",  
  "phone": "01/6129-851",  
  "room": "C11-06"  
}
```

- odgovor: 304 Not Modified
- header: "etag" : "Person with id 100 not found."

Primjer upita i odgovora [7]

- DELETE /persons/2
- odgovor: 204 No Content

Primjer upita i odgovora [8]

- DELETE /persons/100
- odgovor: 404 Not Found

Klasa Person

```
public class Person implements Serializable {  
  
    private final long id;  
    private String firstName, lastName, phone, room;  
  
    public Person(long id, String firstName, String lastName, String phone, String  
room) {  
        this.id = id;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.phone = phone;  
        this.room = room;  
    }  
  
    //getters and setters  
}
```

Klasa PersonFactory

```
public class PersonFactory {  
  
    public static List<Person> generate() {  
        List<Person> list = new LinkedList<>();  
  
        list.add(new Person(0, "Goran", "Delač", "01/6129-549", "D-339-2"));  
        list.add(new Person(1, "Marin", "Šilić", "01/6129-549", "D-339-2"));  
        list.add(new Person(2, "Marin", "Vuković", "01/6129-658", "C07-04"));  
        list.add(new Person(3, "Krešimir", "Pripužić", "01/6129-745", "C08-17"));  
  
        return list;  
    }  
}
```

Klasa PersonService [1]

```
@Service
public class PersonService {

    private final List<Person> persons = PersonFactory.generate();

    public List<Person> get() {
        Collections.sort(persons, Comparator.comparing(Person::getId));
        return persons;
    }

    public void insert(Person person) {
        persons.add(person);
    }
}
```

Klasa PersonService [2]

```
public boolean delete(long id) {  
    Iterator<Person> iterator = persons.iterator();  
  
    while (iterator.hasNext()) {  
        if (iterator.next().getId() == id) {  
            iterator.remove();  
            return true;  
        }  
    }  
  
    return false;  
}
```

Klasa PersonService [3]

```
public boolean update(Person person, long id) {  
    Iterator<Person> iterator = persons.iterator();  
    boolean updated = false;  
    while (iterator.hasNext()) {  
        if (iterator.next().getId() == id) {  
            iterator.remove();  
            updated = true;  
            break;  
        }  
    }  
    if (updated)  
        persons.add(person);  
  
    return updated;  
}
```

Klasa PersonController [1]

```
@RestController  
public class PersonController {  
  
    @Autowired  
    private PersonService personService;  
  
    @GetMapping("/persons")  
    public List<Person> get() {  
        return personService.get();  
    }  
}
```

Klasa PersonController [2]

```
@PostMapping("/persons")
public ResponseEntity insert(@RequestBody Person person) {
    personService.insert(person);

    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.LOCATION,
linkTo(methodOn(PersonController.class).get(person.getId())).toString());
    ResponseEntity response = new ResponseEntity(headers,
HttpStatus.CREATED);
    return response;
}
```

Klasa PersonController [3]

```
@GetMapping("/persons/{id}")
public Person get(@PathVariable("id") long id) {
    Person person = personService.get(id);
    if (person != null) {
        return person;
    } else {
        throw new ResourceNotFoundException("Person
with id " + id + " not found.");
    }
}
```

Klasa PersonController [4]

```
@PutMapping("/persons/{id}")
public ResponseEntity update(@RequestBody Person person,
@PathVariable("id") long id) {
    if (personService.update(person, id)) {
        return ResponseEntity.noContent().build();
    } else {
        return ResponseEntity.status(HttpStatus.NOT_MODIFIED)
            .eTag("Person with id " + id + " not found.")
            .build();
    }
}
```

Klasa PersonController [5]

```
@DeleteMapping(value = "/persons/{id}")
public ResponseEntity delete(@PathVariable("id") long
id) {
    if (personService.delete(id)) {
        return ResponseEntity.noContent().build();
    } else {
        throw new ResourceNotFoundException("Person
with id " + id + " not found.");
    }
}
```

Kako vratiti XML reprezentaciju resursa

- U build.gradle dodati:

```
dependencies {  
    implementation 'com.fasterxml.jackson.dataformat:jackson-  
dataformat-xml'  
}
```

- U klasi PersonsController promijeniti bilješku (*annotation*) @GetMapping:
`@GetMapping(value = "/persons", produces = { "application/json",
"application/xml" })
//@GetMapping("/persons")`
- U zaglavlje HTTP zahtjeva dodati: **Accept application/xml**

Sigurnost RESTful web-usluga

- Moguće je postići značajnu razinu sigurnosti
 - Korištenje protokola HTTPS umjesto HTTP
 - Korištenje autentifikacije korisnika
- Detalji su izvan opsega ovog predavanja

Programska potpora komunikacijskim sustavima

9. predavanje, 17. svibnja 2023.

• doc. dr. sc. Jelena Božek

Protokoli HTTP i TCP



- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Protokol HTTP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom HTTP
- Protokol TCP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom TCP



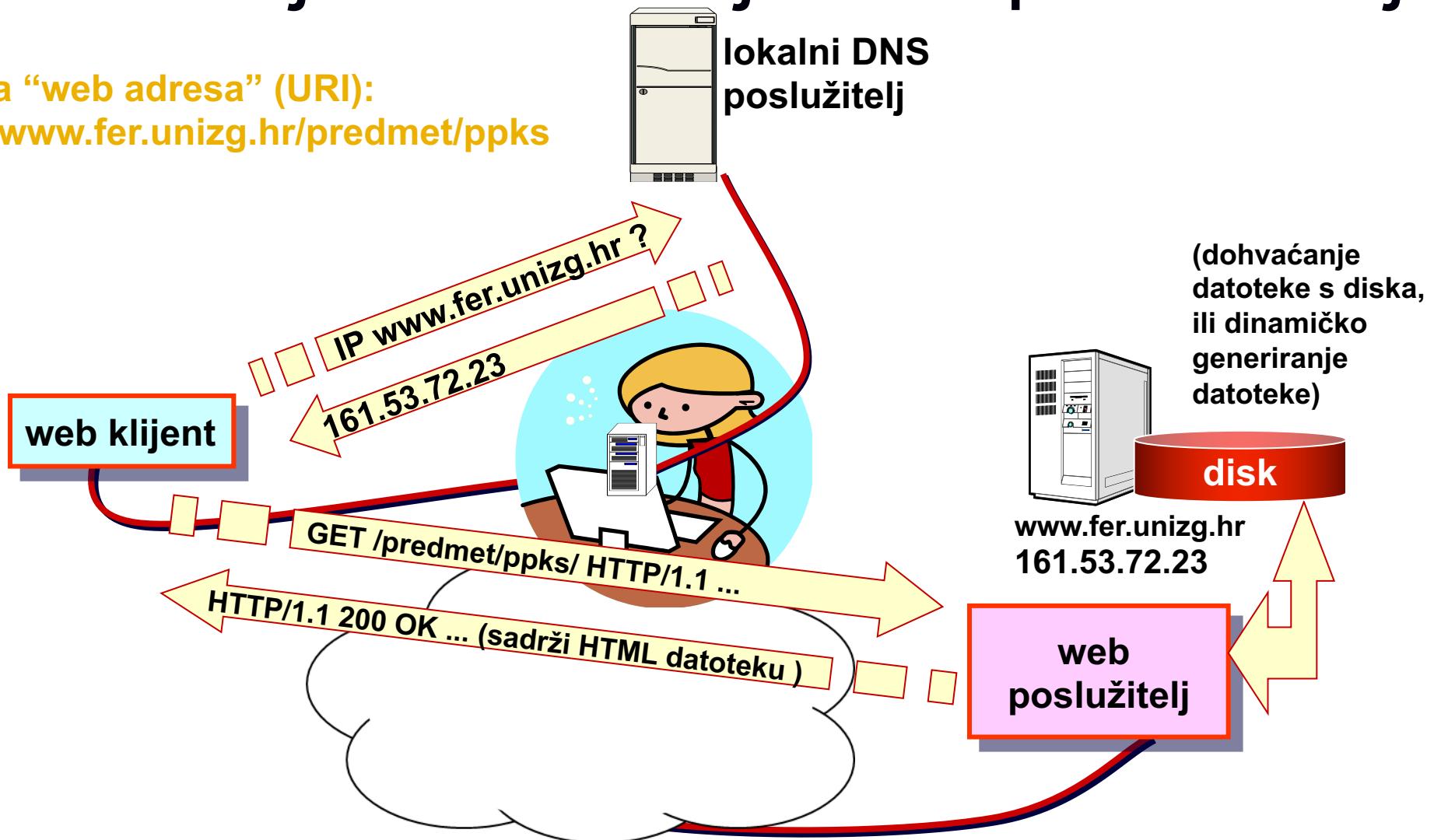
Protokol HTTP

Protokol *Hypertext Transfer Protocol (HTTP)*

- internetski protokol aplikacijskog sloja
- definira format i način razmjene poruka
 - tekstualan zapis
- vrste poruka:
 - **zahtjev** (“metoda” - naziv potječe od terminologije iz područja objektno-orientiranog programiranja)
 - definira operaciju (metodu), resurs, protokol
 - **odgovor** (ishod zahtjeva i rezultat)
 - ishod zahtjeva (uspjeh, neuspjeh, greška,...) opisan statusnim kôdom
 - za neke vrste zahtjeva, kao rezultat uspješnog ishoda, u tijelu odgovora dostavlja se sadržaj zatraženog resursa

Komunikacija HTTP klijenta i poslužitelja

Odabrana "web adresa" (URI):
<http://www.fer.unizg.hr/predmet/ppks>



Specifikacije protokola HTTP [1]

- prva verzija HTTP/0.9 - ograničene mogućnosti
 - podržava prijenos samo hipertekstualnih dokumenata
- **HTTP/1.0** - prvi (*Informational*) RFC (RFC 1945), 1996.
 - podržava prijenos različitih tipova podataka
 - posuđuje koncepte iz *media type* standarda
 - zadržava kompatibilnost unazad
 - sredinom 90-tih - HTTP promet dominira - HTTP/1.0 neučinkovit
 - svako sjedište mora biti na drugom poslužitelju
 - preko jedne konekcije ostvaruje se jedan HTTP zahtjev
 - nema podrške za upravljanje performansama - *cache*, *proxy*, djelomični dohvati
 - WWW prozvan "World Wide Wait"

Specifikacije protokola HTTP [2]

- **HTTP/1.1** - RFC 2616, lipanj 1999.
 - zadržava kompatibilnost unazad prema HTTP 1.1
 - RFC 2617 - autentifikacija i sigurnost
 - poboljšanja:
 - jedno fizičko računalo - više Web poslužitelja - "virtualni host"
 - trajne konekcije - više zahtjeva preko jedne TCP konekcije
 - djelomični dohvat sadržaja
 - bolja podrška za priručna spremišta (*cache*) i zastupnike(*proxy*)
 - pregovaranje o sadržaju datoteke (*content negotiation*)
 - bolji sigurnosni mehanizmi – autentifikacija
- Postoje i novije verzije: HTTP/2 (2015. godine) i HTTP/3 (predloženi standard, lipanj 2022.)

HTTP poruke

- zahtjev i odgovor moraju biti ispravno formatirani
- HTTP definira opći format poruke
 - tekstualan zapis
 - naslanja se na format e-mail poruke (RFC 822) i *media type* standarda
 - dijele neka načela, ali ne sasvim i ne potpuno
 - npr. ne koriste se sva *media type* zaglavlja
 - npr. tijelo ne mora biti 7-bitni ASCII

Poruke protokola HTTP

zahtjev

GET /predmet/ppks HTTP/1.1
Host: www.fer.hr

...

Accept-Language: hr, en
Accept-Encoding: gzip,deflate

...

odgovor

HTTP/1.1 200 OK
Date: Mon, 07 Apr 2022 17:31:09 GMT
Server: Apache/2.2.8 (FreeBSD) ..
Last-Modified: Mon, 30 Jan 2022
16:12:36 GMT

...

Keep-Alive: timeout=3, max=61
Connection: Keep-Alive
Content-Type: application/javascript

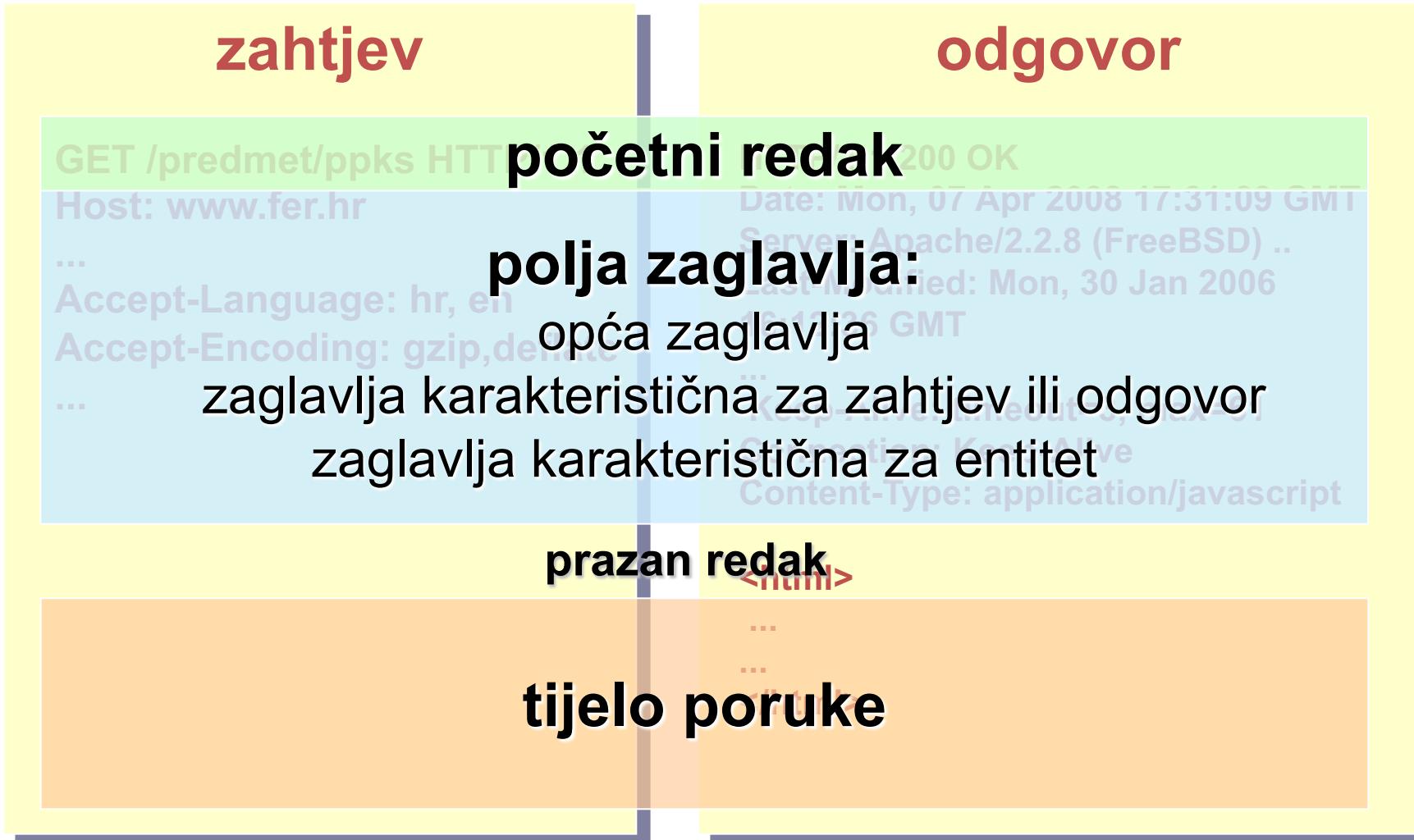
<html>

...

...

</html>

Format poruka



Oblikovanje zahtjeva

- početni redak sadrži (*request line*):
 - resurs nad kojim je podnesen zahtjev
 - metodu (operacija) koja se traži nad tim resursom
 - verziju protokola koju koristi
- primjeri:
 - **GET / HTTP/1.0**
 - **POST /shop/order HTTP/1.1**
 - **HEAD /search?q=raspored HTTP/1.0**

Metode zahtjeva

- metoda zahtjeva određuje što se traži od resursa
- HTTP/1.1 definira 8 metoda i omogućuje dodavanje novih metoda (*extensions*):
 - OPTIONS
 - **GET**
 - HEAD
 - **POST**
 - **PUT**
 - **DELETE**
 - TRACE
 - CONNECT
- važna svojstva metoda: sigurna (*safe*), indempotentna (*idempotent*), pamtljiv odgovor (*cachable*)

Zaglavlj zahtjeva

METHOD:

- GET
- POST
- HEAD
- PUT
- OPTIONS
- TRACE
- CONNECT
- DELETE

Method Request-URI HTTP-Version

parametar1: vrijednost
parametar2: vrijednost
parametar3: vrijednost
....

prazni redak>

HTTP/1.0
HTTP/1.1

general_header

Cache-Control:
Connection:
Date:
Pragma:
Trailer:
Transfer-Encoding:
Upgrade:
Warning:

request_header

Accept: If-Modified-Since:
Accept-Charset: If-None-Match:
Accept-Encoding: If-Range:
Accept-Language: If-Unmodified-Since:
Authorization:
Expect:
From:
Host:
If-Match:

entity_header

Allow:
Content-Encoding:
Content-Language:
Content-Length:
Content-Location:
Content-MD5:
Content-Range:
Content-Type:
Expires:
Last-Modified:
extension-header

Odgovor poslužitelja

- početni redak sadrži:
 - verziju protokola
 - statusni kôd
 - opisnu frazu
- u **tijelu** odgovora se obično prenosi reprezentacija resursa (“entitet”) koju preglednik treba prikazati korisniku
- neka polja zaglavlja:
 - **Content-Type**: format entiteta
 - **Content-Length**: duljina entiteta u tijelu u oktetima

HTTP/1.1 303 See Other

HTTP/1.1 200 OK

HTTP/1.1 404 Not Found

Statusni kôd odgovora

- sastoji se od tri dekadske znamenke
- pet kategorija:
 - 1xx – **Informativne** - ne naznačuju ni uspjeh, ni neuspjeh
 - 2xx – **Uspjeh** - poslužitelj je primio, razumio i ispunio zahtjev
 - 3xx – **Preusmjeravanje** - potrebno poduzeti dodatne akcije
 - 4xx – Greška na **klijentu** - zahtjev je neispravan
 - 5xx – Greška na **poslužitelju** - zahtjev je ispravan, ali poslužitelj ga ne može ispuniti

Primjer HTTP klijenta

- Napravit ćemo klijenta za REST poslužitelja s prethodnog predavanja korištenjem klase `java.net.http.HttpClient`
- Implementirat ćemo samo HTTP metodu GET
 - `GET /persons` treba dohvatiti listu osoba
 - `GET /persons/4` treba dohvatiti osobu čiji je ID jednak 4

Proširenje klase Person

```
...  
import com.google.gson.Gson;  
  
public class Person {  
  
    private static final Gson gson = new Gson();  
  
    ...  
  
    public String toJson() {  
        return gson.toJson(this);  
    }  
  
    public static Person fromJson(String json) {  
        return (Person) gson.fromJson(json, Person.class);  
    }  
  
    public static String listToJson(List<Person> list) {  
        return gson.toJson(list);  
    }  
  
    public static List<Person> listFromJson(String json) {  
        Type listOfPerson = new TypeToken<List<Person>>() {}.getType();  
        List<Person> list = gson.fromJson(json, listOfPerson);  
        return list;  
    }  
}
```



Klasa PersonHTTPGetClient [1]

```
public class PersonHTTPGetClient implements PersonInterface {

    private final String schemeHostPort;
    private final HttpClient httpClient;

    public PersonHTTPGetClient(String host, int port) {
        schemeHostPort = "http://" + host + ":" + port;
        httpClient = HttpClient.newBuilder().build();
    }

    public static void main(String[] args) throws Exception {
        PersonHTTPClient personHTTPClient = new PersonHTTPClient("localhost", 8000);

        //read resources
        List<Person> list = personHTTPClient.get();
        System.out.println(list);
        System.out.println("");

        ...
    }
}
```

Klasa PersonsHTTPGetClient [2]

```
@Override
public List<Person> get() {
    try {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(schemeHostPort + "/persons"))
            .GET()
            .build();

        HttpResponse response = httpClient.send(request, BodyHandlers.ofString());
        System.out.println("Read all persons:");
        System.out.println(response);
        System.out.println(response.body());
        List<Person> list = Person.listFromJson(response.body().toString());
        return list;
    } catch (IOException | InterruptedException ex) {
        return null;
    }
}
```

Klasa PersonsHTTPGetClient [3]

```
@Override
public Person get(long id) {
    try {
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(schemeHostPort + "/persons/" + id))
            .GET()
            .build();

        HttpResponse response = httpClient.send(request, BodyHandlers.ofString());
        System.out.println("Read person with id=" + id + ":");
        System.out.println(response);
        System.out.println(response.body());
        Person person = (Person) Person.fromJson(response.body().toString());
        return (response.statusCode() == 200) ? person : null;
    } catch (IOException | InterruptedException ex) {
        return null;
    }
}
```

Klasa PersonsHTTPClient

- Ova klasa je proširenje klase PersonsHTTPGetClient
- Predstavlja potpuno implementiranog klijenta za rad s REST-uslugom s prethodnog predavanja
 - Sadrži sve metode za upravljanje osobama kao resursima, a ne samo metode za dohvaćanje resursa (kao kod klase PersonsHTTPGetClient)
- Testira ispravan rad aplikacije s prethodnog predavanja

Primjer HTTP poslužitelja

- Napravit ćemo REST poslužitelja s prethodnog predavanja korištenjem klase com.sun.net.httpserver.HttpServer
- Implementirat ćemo samo HTTP metodu GET
 - GET /persons treba vratiti listu osoba
 - GET /persons/4 treba vratiti osobu čiji je ID jednak 4

Klasa PersonsHTTPGetServer

```
public class PersonsHTTPGetServer {  
  
    private final PersonService personService;  
    private final HttpServer httpServer;  
  
    public PersonsHTTPGetServer(int port) throws IOException {  
        personService = new PersonService();  
        httpServer = HttpServer.create(new InetSocketAddress(port), 0);  
        httpServer.createContext("/persons", new PersonsHTTPGetServer.PersonsHandler());  
    }  
  
    public void start() {  
        httpServer.start();  
    }  
  
    public static void main(String[] args) throws IOException {  
        PersonsHTTPGetServer server = new PersonsHTTPGetServer(8000);  
        server.start();  
    }  
}
```

Unutarnja klasa PersonsHandler [1]

```
class PersonsHandler implements HttpHandler {  
  
    @Override  
    public void handle(HttpExchange httpExchange) throws IOException {  
        switch (httpExchange.getRequestMethod()) {  
            case "GET":  
                handleGet(httpExchange);  
                break;  
            case "POST":  
                handlePost(httpExchange);  
                break;  
            case "PUT":  
                handlePut(httpExchange);  
                break;  
            case "DELETE":  
                handleDelete(httpExchange);  
                break;  
        }  
    }  
}
```

Unutarnja klasa PersonsHandler [2]

```
private void handleGet(HttpExchange httpExchange) throws IOException {
    URI uri = httpExchange.getRequestURI();
    byte[] responseBody = new byte[0]; //default body for malformed requests
    int responseCode = 404; //default status code for malformed requests

    if (uri.getPath().equals("/persons")) {//return all persons
        responseBody = Person.listToJson(personService.get()).getBytes("UTF-8");
        responseCode = 200;
    } else if (uri.getPath().matches("/persons/[0-9]*")) {//return person with given id
        String[] split = uri.getPath().split("/");
        Person person = personService.get(Integer.parseInt(split[2]));
        if (person != null) {
            responseBody = person.toJson().getBytes("UTF-8");
            responseCode = 200;
        }
    }
    httpExchange.getResponseHeaders().set("content-type", "application/json");
    httpExchange.sendResponseHeaders(responseCode, responseBody.length);
    try (OutputStream os = httpExchange.getResponseBody()) { //write and close the response body
        os.write(responseBody);
    }
}
```

Unutarnja klasa PersonsHandler [3]

```
private void handlePost(HttpExchange httpExchange) throws IOException {
    //return status code 501 - no implemented
    httpExchange.sendResponseHeaders(501, -1);
}

private void handlePut(HttpExchange httpExchange) throws IOException {
    //return status code 501 - no implemented
    httpExchange.sendResponseHeaders(501, -1);
}

private void handleDelete(HttpExchange httpExchange) throws IOException {
    //return status code 501 - no implemented
    httpExchange.sendResponseHeaders(501, -1);
}
```



Protokol TCP

Obilježja komunikacije [1]

■ **konekcijska**

- procesi eksplicitno kreiraju konekciju prije razmjene podataka, postoje kontrolne poruke za uspostavu konekcije

■ **bezkonekcijska**

- sve poruke prenose podatke, nema kontrolnih poruka za uspostavu konekcije među procesima

• **perzistentna komunikacija**

- garantira isporuku poruke, poruka se pohranjuje u sustavu i isporučuje primatelju kada je to moguće

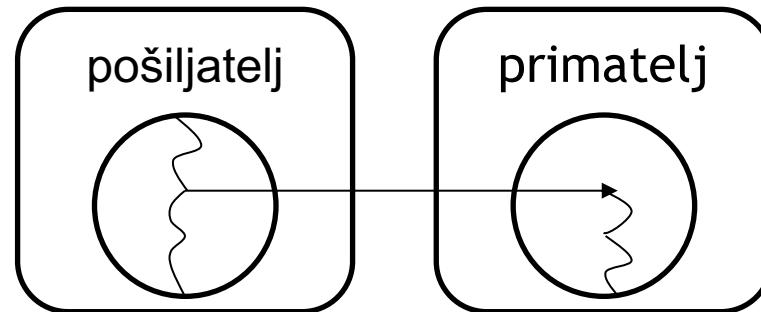
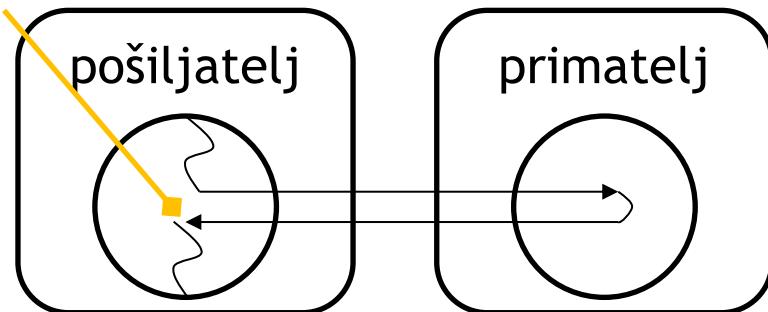
• **tranzijentna komunikacija**

- nepouzdana, garantira isporuku poruke samo ako su pošiljatelj i primatelj poruke istovremeno dostupni

Obilježja komunikacije [2]

- **sinkrona komunikacija**
 - blokira pošiljatelja do primitka potvrde od strane primatelja
- **asinkrona komunikacija**
 - omogućuje pošiljatelju nastavak obrade odmah nakon slanja poruke

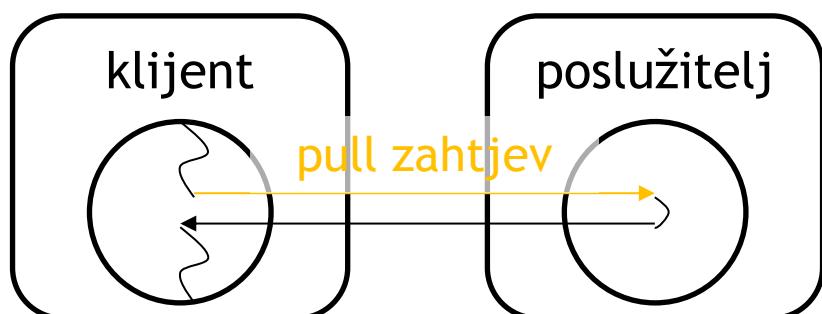
blokiranje



Obilježja komunikacije [3]

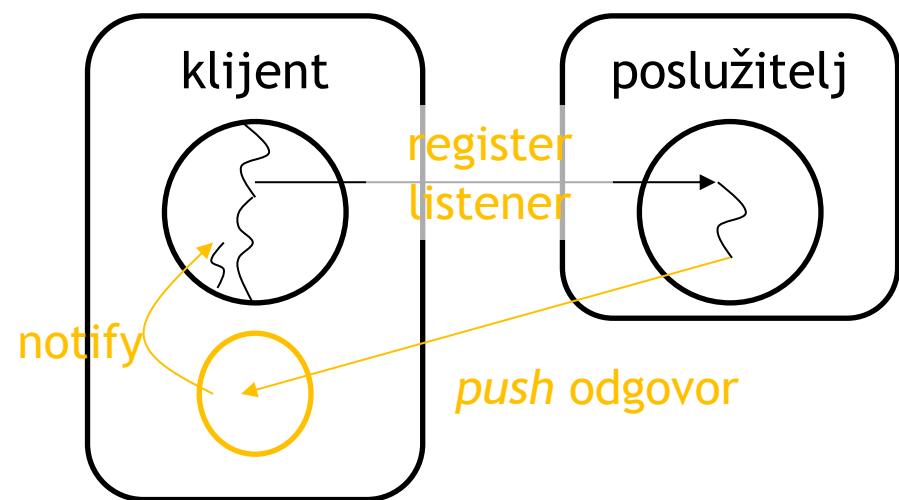
- *pull*

- “klasični” model zahtjev-odgovor



- *push*

- klijent registrira zahtjev i “sluša” odgovor, poslužitelj šalje odgovor nakon što završi obradu zahtjeva



Obilježja komunikacije protokolom TCP

- vremenska ovisnost
 - klijent i poslužitelj moraju biti istovremeno dostupni
- tranzijentna komunikacija
- konekcijska komunikacija
- sinkrona komunikacija
 - klijent šalje zahtjev za kreiranje konekcije i proces je blokiran do uspostave konekcije
- pokretanje komunikacije na načelu *pull*
 - klijent mora znati identifikator poslužitelja

Sloj raspodijeljenog sustava za komunikaciju među procesima

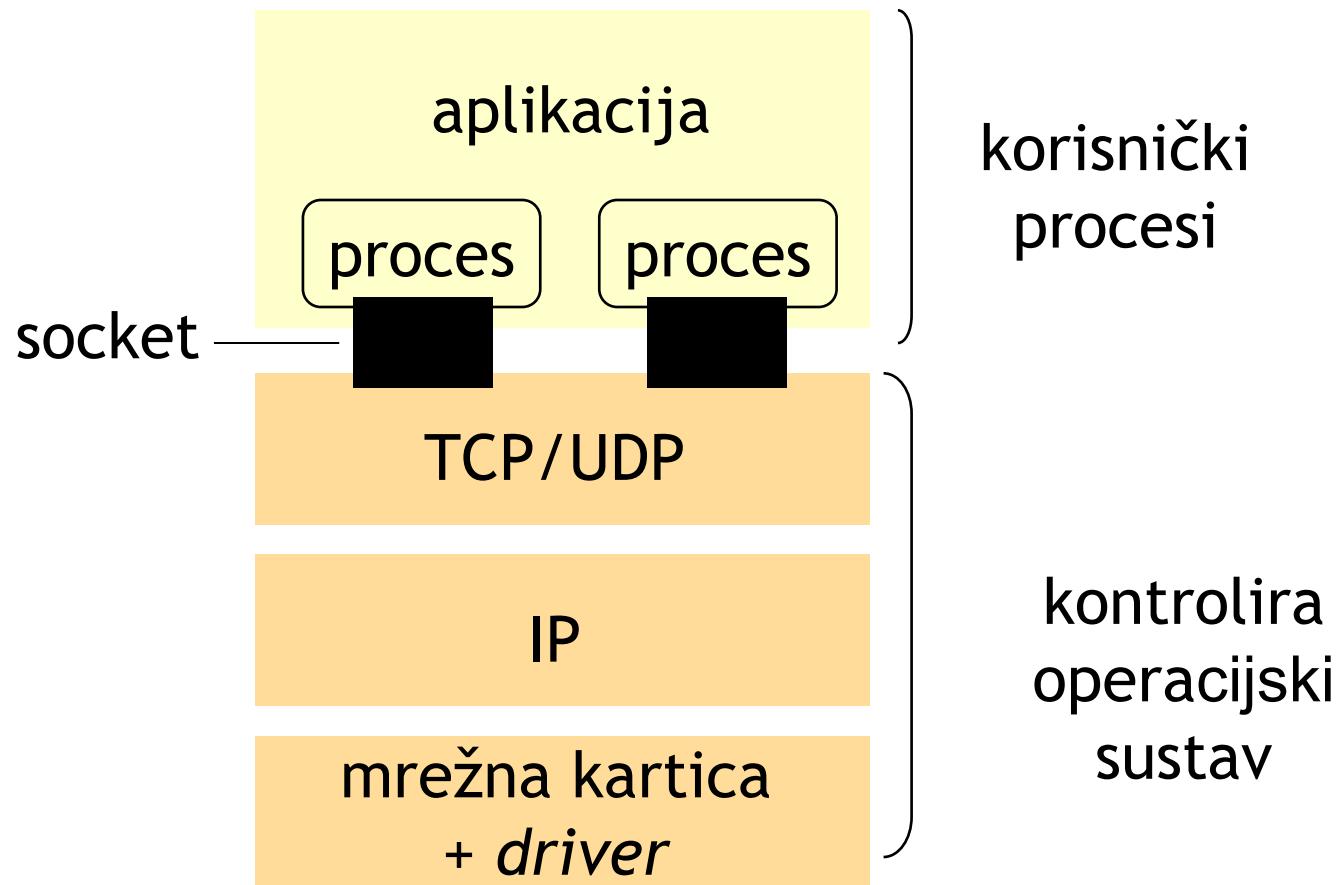
- Postojeća rješenja za komunikaciju raspodijeljenih procesa
 1. komunikacija korištenjem priključnica (*socket API*)
 2. poziv udaljene procedure (*Remote Procedure Call, RPC*)
 3. raspodijeljeni objekti - poziv udaljene metode (*Remote Method Invocation, RMI*)
 4. komunikacija razmjenom poruka (*message-oriented interaction*)
 5. model objavi-preplati (*publish/subscribe*)
 6. REST (*REpresentational State Transfer*)
 7. SOAP (*Simple Object Access Protocol*)

Komunikacija korištenjem priključnica

Socket API

- koristi funkcionalnost transportnog sloja
 - TCP – konekcijski protokol, pouzdan prijenos podataka
 - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (*socket*)
 - pristupna točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
 - viši nivo apstrakcije nad komunikacijskom točkom koju operacijski sustav koristi za pristup transportnom sloju
 - veže se uz vrata (*port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

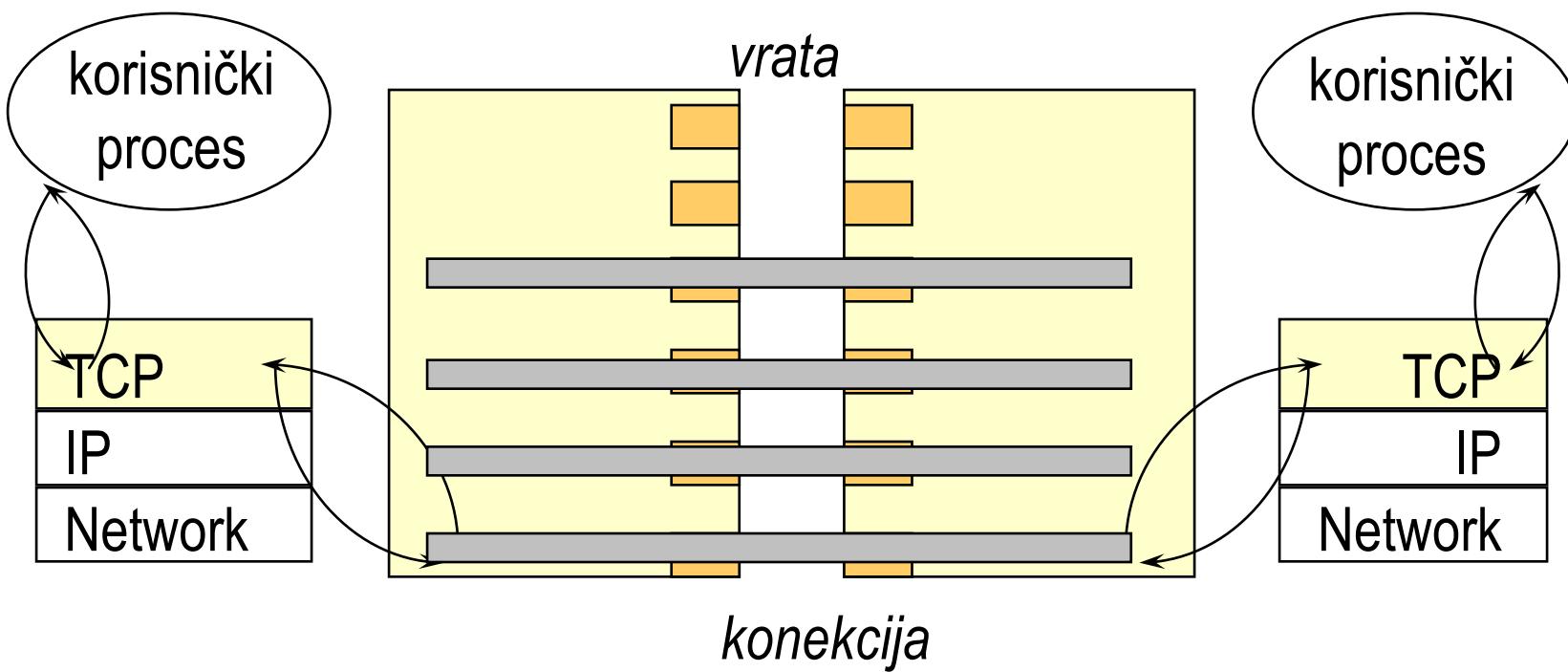
Komunikacija pomoću priključnica



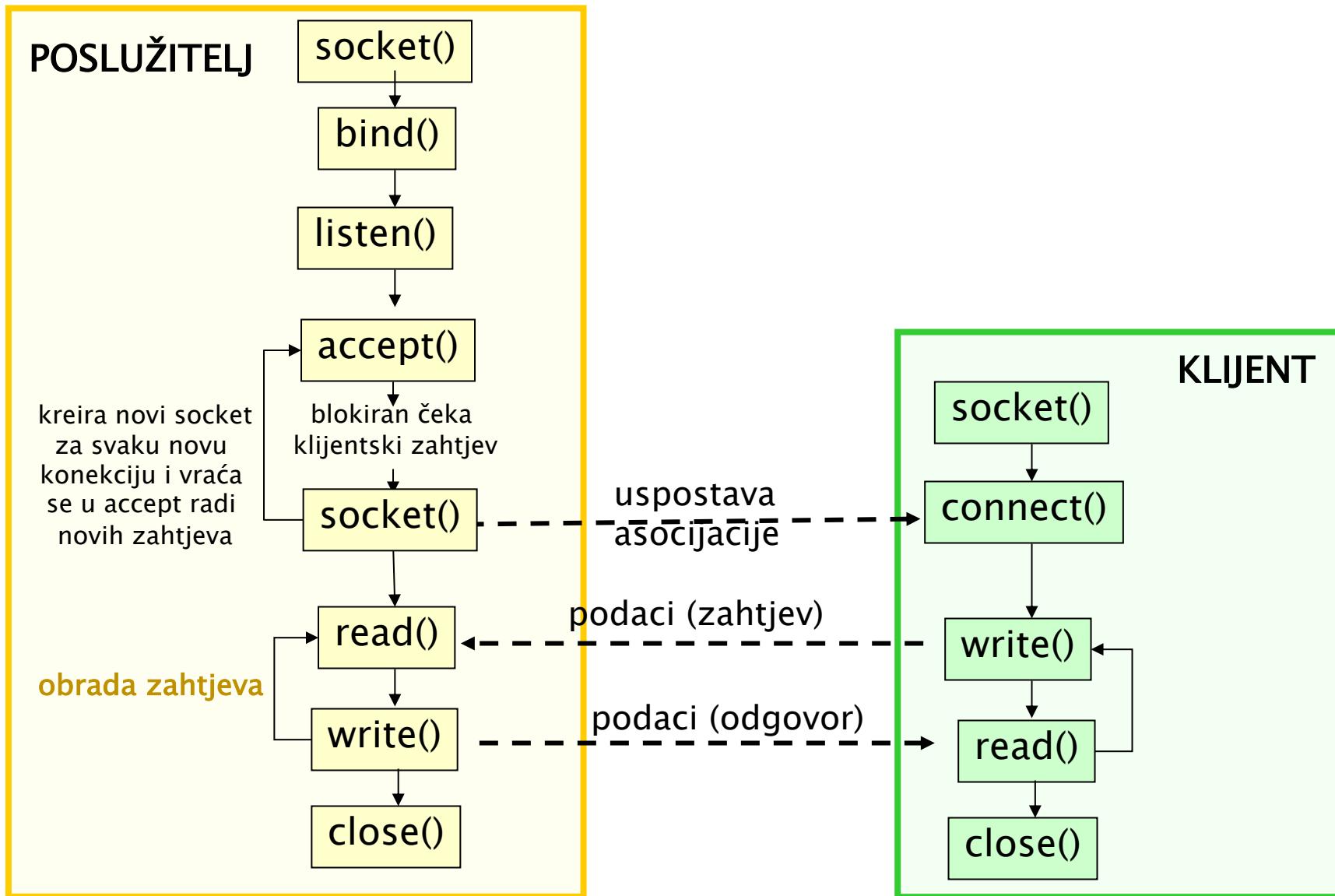
Transportni protocol TCP

Transmission Control Protocol (TCP)

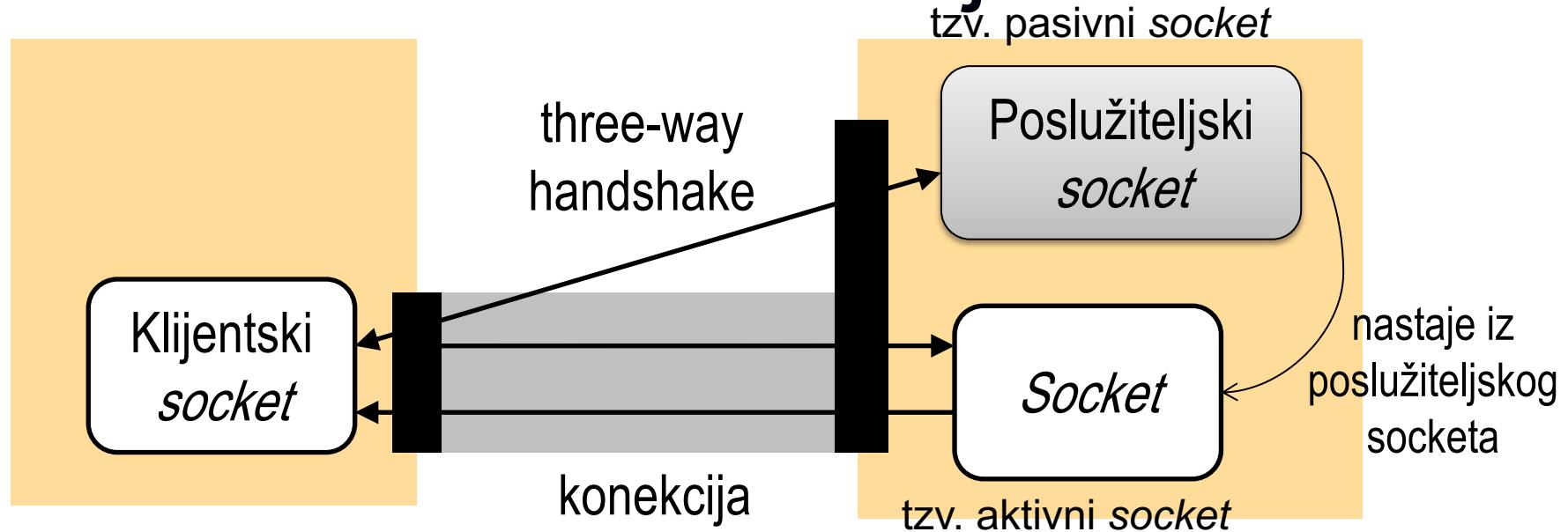
- konekcija između dvije krajnje točke koje se moraju dogovoriti o uspostavi konekcije



Konekcijska komunikacija pomoću socketa TCP



Konkurentni korisnički zahtjevi



- ◆ za svaki novi korisnički zahtjev kreira se **novi socket** (s dva *buffer-a, in i out*) **koji se veže uz konekciju** <poslužiteljska IP adresa i broj vrata (broj vrata ostaje isti kao za poslužiteljski socket), klijentska IP adresa i broj vrata>
- ◆ originalni poslužiteljski socket mora konstantno biti u stanju “osluškivanja”

TCP: implementacija klijenta

1. Kreirati klijentski socket:

```
clientSocket = new Socket( address, port );
```

2. Kreirati I/O stream za komunikaciju s poslužiteljem:

```
InputStream is = clientSocket.getInputStream();
OutputStream os = clientSocket.getOutputStream();
```

3. Komunikacija s poslužiteljem:

```
//Receive byte[] b from server:
int length = is.read(b);
```

```
//Send byte[] b to server:
os.write(b);
```

4. Zatvoriti socket:

```
clientSocket.close();
```

TCP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
ServerSocket serverSocket;  
serverSocket = new ServerSocket( PORT );
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog socketa:

```
Socket clientSocket = serverSocket.accept();
```

3. Kreirati I/O stream za komunikaciju s klijentom

```
InputStream is = clientSocket.getInputStream();  
OutputStream os = clientSocket.getOutputStream();
```

4. Komunikacija s klijentom

5. Zatvoriti kopiju socketa:

```
clientSocket.close();
```

6. Zatvoriti poslužiteljski socket:

```
serverSocket.close();
```

Primjer TCP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa TCPClient [1]

```
public class TCPClient {  
    public static void main(String[] args) {  
        try ( Socket clientSocket = new Socket("localhost", 12345); //SOCKET -> CONNECT  
              BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(  
clientSocket.getOutputStream(), "UTF-8")); //  
              BufferedReader reader = new BufferedReader(new InputStreamReader(  
clientSocket.getInputStream(), "UTF-8"))) {  
  
            //start sending readings - NA SLJEDEĆEM SLAJDU!!!  
            while (true) {  
                ...  
            }  
        } catch (IOException ex) {  
            Logger.getLogger(TCPClient.class.getName()).log(Level.SEVERE, null, ex);  
        } //CLOSE  
    }  
}
```

Klasa TCPClient [2]

```
//start sending readings
while (true) {
    //send a reading
    Reading reading = new Reading("id_5", "temperature", (Math.random() * 60) - 20, "C");
    writer.write(reading.toJson()); writer.newLine(); //WRITE
    writer.flush();
    System.out.println("Sent: " + reading);

    //receive confirmation
    String confirmation = reader.readLine(); //READ
    System.out.println("Received: " + confirmation);

    try {
        Thread.sleep(5000);
    } catch (InterruptedException ex) {
        //do nothing
    }
}
```

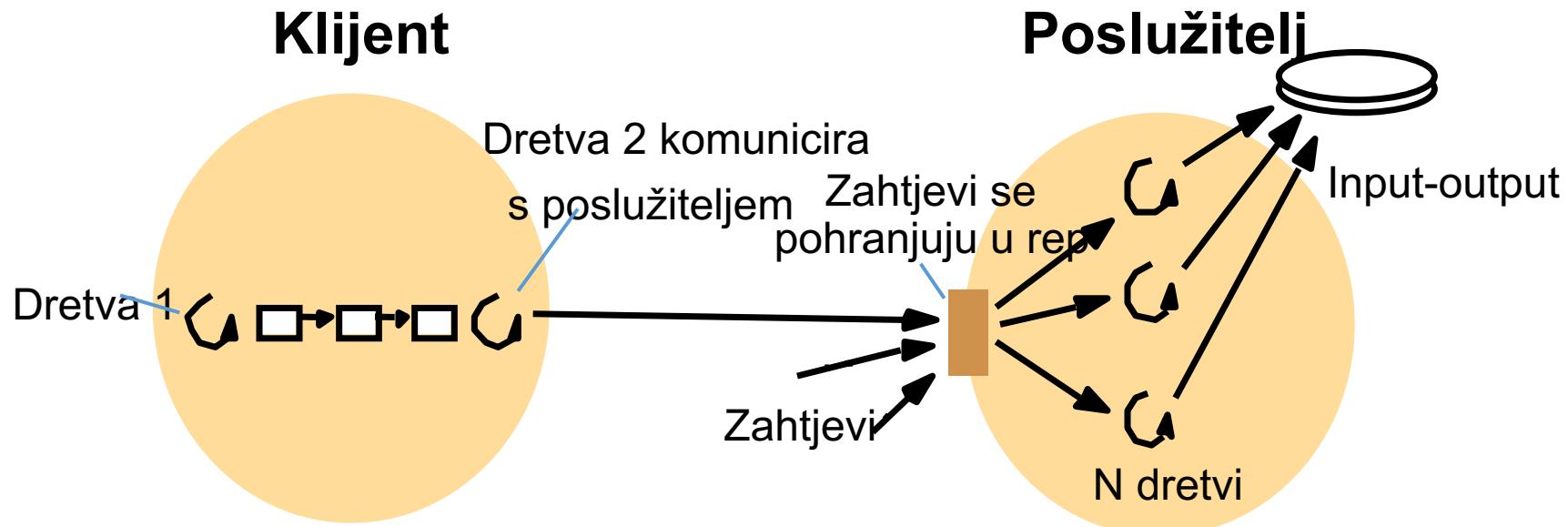
Klasa SingleClientTCPServer [1]

```
public class SingleClientTCPServer {  
    private int port;  
  
    public SingleClientTCPServer(int port) {  
        this.port = port;  
    }  
  
    public static void main(String[] args) {  
        SingleClientTCPServer server = new SingleClientTCPServer(12345);  
        server.start();  
    }  
}
```

Klasa SingleClientTCPServer [2]

```
public void start() {  
    try ( ServerSocket serverSocket = new ServerSocket(port); //SOCKET -> BIND -> LISTEN  
          Socket clientSocket = serverSocket.accept(); //ACCEPT -> SOCKET  
          BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), "UTF-8")); //  
          BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream(), "UTF-8"))) {  
  
        //start consuming readings  
        String jsonReading;  
        while ((jsonReading = reader.readLine()) != null) { //READ  
            //receive a reading  
            Reading reading = Reading.fromJson(jsonReading);  
            System.out.println("Received: " + reading);  
  
            //send a confirmation  
            String confirmation = Messages.CONFIRM;  
            writer.write(confirmation); writer.newLine(); writer.flush(); //WRITE  
            System.out.println("Sent: " + confirmation);  
        }  
    } catch (IOException ex) {  
        ex.printStackTrace();  
    } //CLOSE  
}
```

Višedretveni poslužitelj i klijenti



Uobičajene zadaće na strani klijenta:

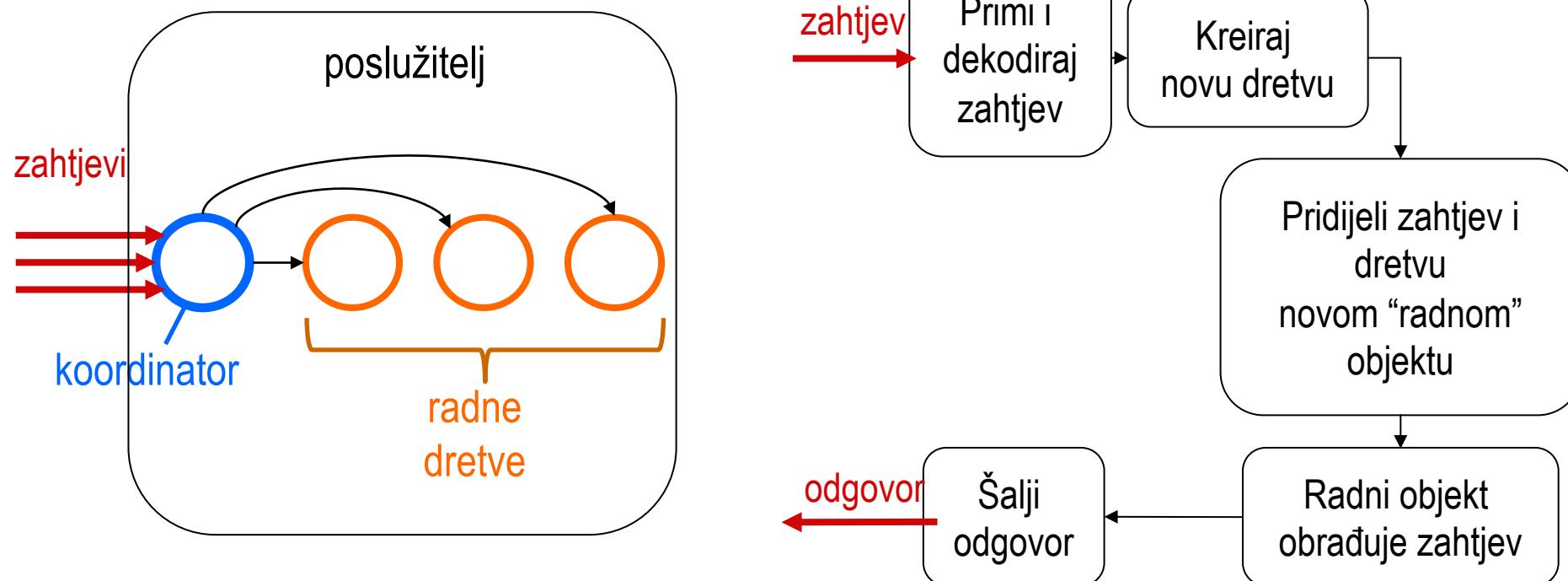
- korisničko sučelje,
- otvaranje mrežne konekcije i primanje podataka

Uobičajene zadaće na strani poslužitelja:

- primanje konkurentnih klijentskih zahtjeva
- složena obrada podataka
- rad s diskom/bazom podataka

Višedretveni poslužitelj

Model koordinator/radna dretva (*dispatcher/worker model*)



Klasa TCPServer [1]

```
public class TCPServer {  
    private int port;  
  
    public TCPServer(int port) {  
        this.port = port;  
    }  
  
    public void start() {  
        try ( ServerSocket serverSocket = new ServerSocket(port)) { //SOCKET -> BIND -> LISTEN  
            while (true) {  
                try {  
                    Socket clientSocket = serverSocket.accept(); //ACCEPT -> SOCKET  
                    new Thread(new ServerTask(clientSocket)).start();  
                } catch (IOException ex) {  
                    ex.printStackTrace();  
                }  
            }  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        } //CLOSE  
    }  
  
    public static void main(String[] args) {  
        TCPServer server = new TCPServer(12345);  
        server.start();  
    }  
}
```

Klasa TCPServer [2]

```
private class ServerTask implements Runnable {  
    private final Socket clientSocket;  
  
    public ServerTask(Socket clientSocket) {  
        this.clientSocket = clientSocket;  
    }  
    @Override  
    public void run() {  
        try ( BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), "UTF-8")); //  
              BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream(), "UTF-8"))) {  
            //start consuming readings  
            String jsonReading;  
            while ((jsonReading = reader.readLine()) != null) { //READ  
                //receive a reading  
                Reading reading = Reading.fromJson(jsonReading);  
                System.out.println("Received: " + reading);  
                //send confirmation  
                String confirmation = Messages.CONFIRM;  
                writer.write(confirmation); writer.newLine(); writer.flush(); //WRITE  
                System.out.println("Sent: " + confirmation);  
            }  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

Programska potpora komunikacijskim sustavima

10. predavanje, 24. svibnja 2023.

• doc. dr. sc. Jelena Božek

Protokoli UDP i SCTP



- Predavanja izradio prof. dr. sc. Krešimir Pripužić, 2022.
- Predavanja doradila doc. dr. sc. Jelena Božek, 2023.

Sadržaj predavanja

- Protokol UDP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom UDP
- Protokol SCTP
- Primjer klijenta i poslužitelja koji komuniciraju protokolom SCTP



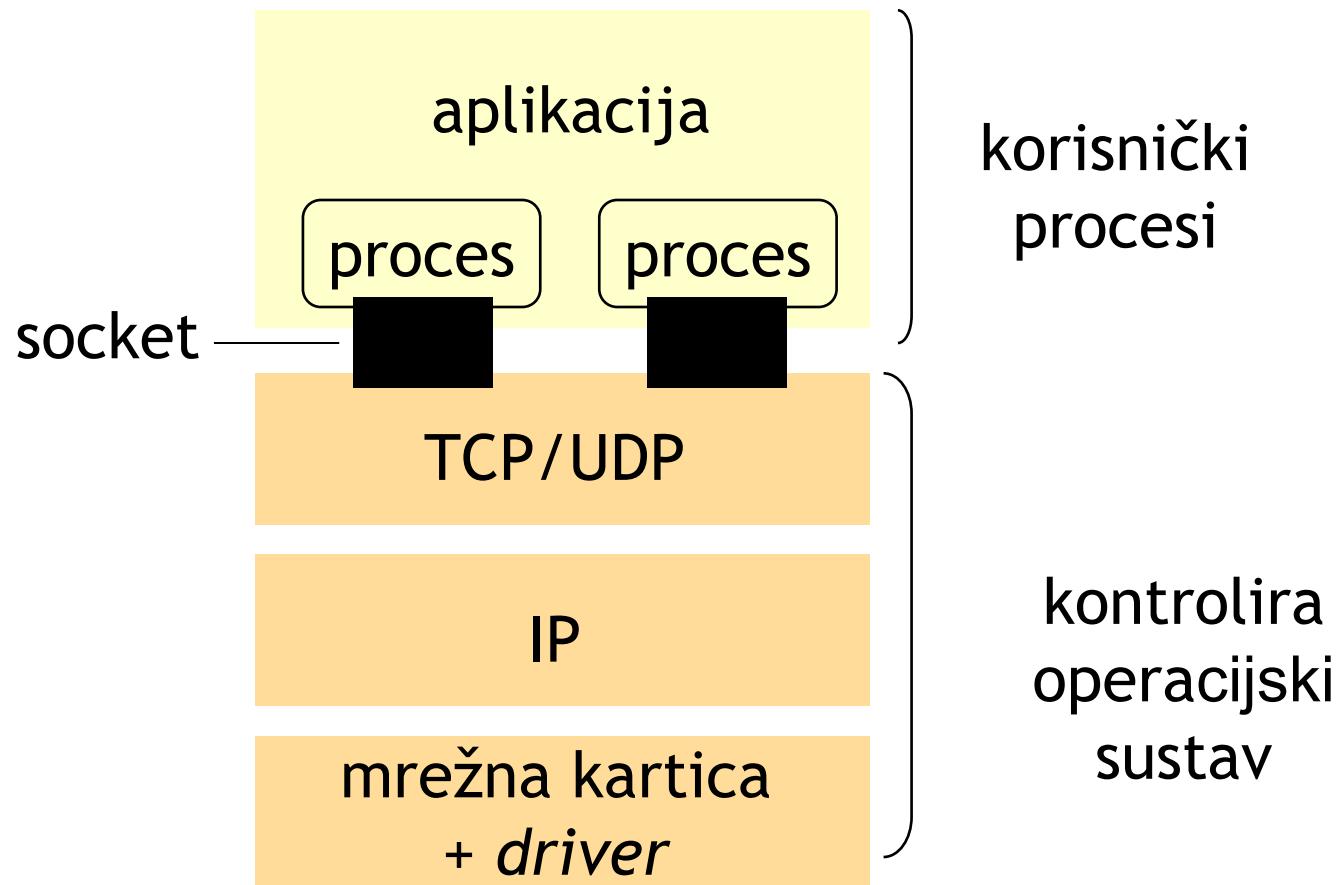
Protokol UDP

Komunikacija korištenjem priključnica

Socket API

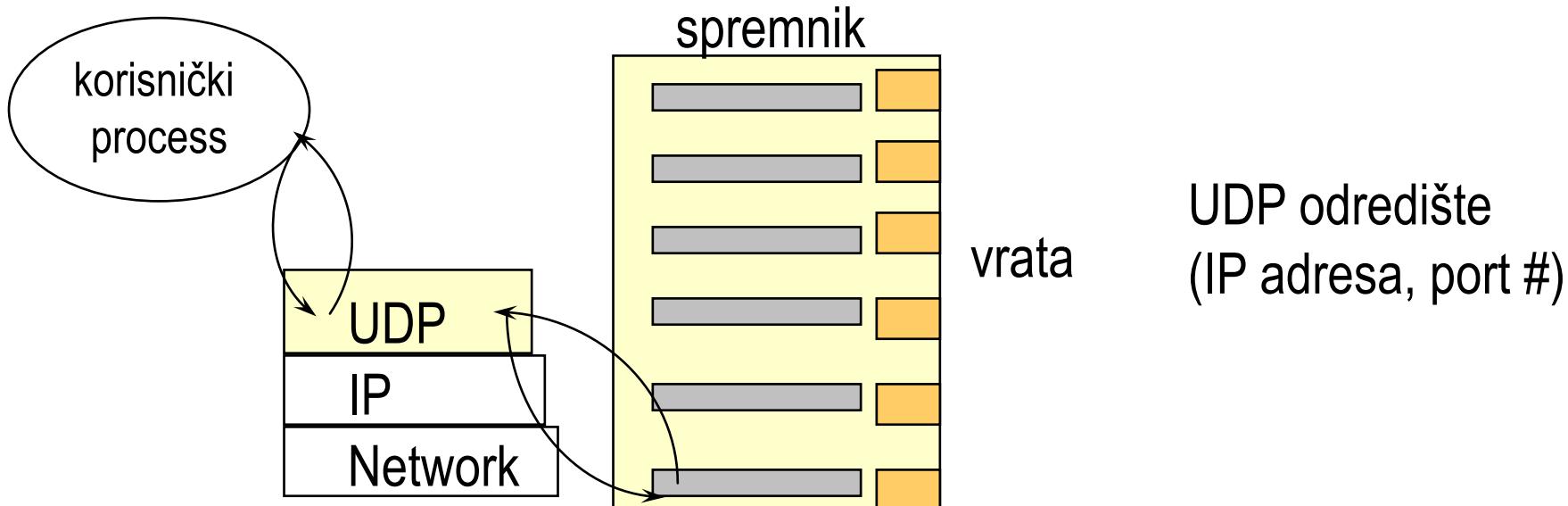
- koristi funkcionalnost transportnog sloja
 - TCP – konekcijski protokol, pouzdan prijenos podataka
 - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (*socket*)
 - pristupna točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
 - viši nivo apstrakcije nad komunikacijskom točkom koju operacijski sustav koristi za pristup transportnom sloju
 - veže se uz vrata (*port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

Komunikacija pomoću priključnica

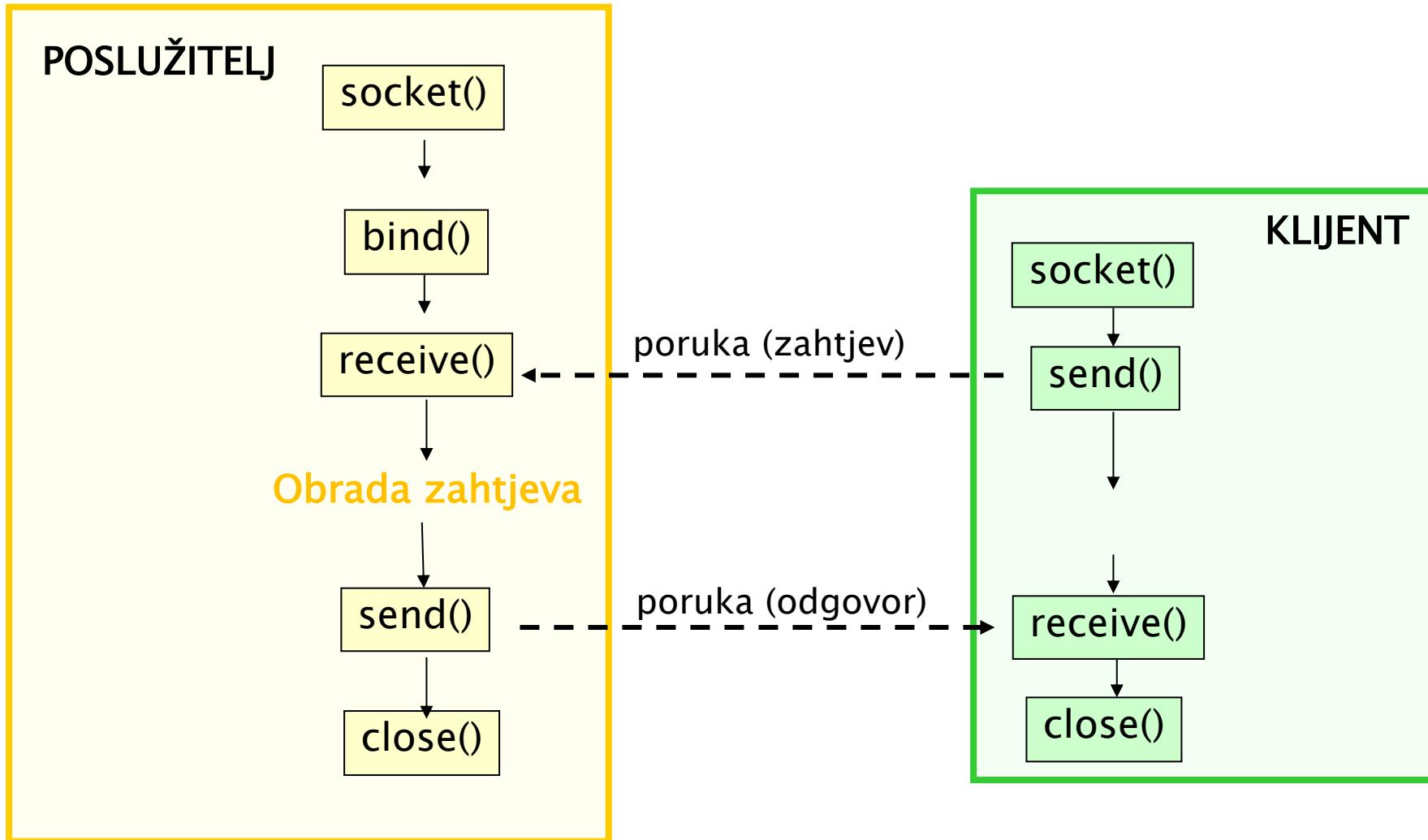


Transportni protokol UDP

- User Datagram Protocol (UDP)
 - komunikacija se odvija preko vrata (engl. portova) koje dodjeljuje operacijski sustav na strani klijenta, na strani poslužitelja se koriste "dobro poznata vrata"



Komunikacija pomoću socketa *UDP*



Obilježja socketa UDP

- model klijent-poslužitelj
- vremenska ovisnost procesa
 - poslužitelj mora biti aktivan za primanje datagrama
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- asinkrona komunikacija
 - klijent šalje datagram i nastavlja obradu, nema blokiranja pošiljatelja
- nepouzdana komunikacija
- može se koristiti za implementaciju komunikacije na načelu *pull* i *push*

UDP: implementacija klijenta

1. Kreirati socket:

```
DatagramSocket socket = new DatagramSocket();
```

2. Kreirati paket i napuniti ga podacima:

```
byte[] sndBuffer = new byte[256];
```

```
DatagramPacket packet = new DatagramPacket(sndBuffer,  
    sndBuffer.length, dstAddress, dstPort);
```

3. Slanje paketa:

```
socket.send( packet );
```

4. Po potrebi obrada i čekanje odgovora

5. Zatvoriti socket:

```
socket.close();
```

UDP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
DatagramSocket socket = new DatagramSocket( PORT );
```

2. Kreirati paket (prazan, priprema za primanje):

```
byte[] rcvBuffer = new byte[256];
DatagramPacket packet = new DatagramPacket(rcvBuffer,
rcvBuffer.length);
```

3. Čekati korisnički paket (blokira proces do klijentskog zahtjeva!):

```
socket.receive(packet);
```

4. Obrada pristiglog paketa i po potrebi odgovor klijentu

5. Zatvoriti socket (gasi poslužitelja):

```
serverSocket.close();
```

Primjer UDP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa UDPClient [1]

```
public class UDPClient {  
    public static void main(String[] args) {  
        try ( DatagramSocket socket = new DatagramSocket() ) { //SOCKET  
            while (true) {  
                //send a reading  
                Reading reading = new Reading("id_5", "temperature", (Math.random() * 60) - 20, "C");  
                byte[] buffer = reading.toJson().getBytes();  
  
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length,  
InetAddress.getByName("localhost"), 11111);  
                socket.send(packet); //SEND  
                System.out.println("Sent: " + reading);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Klasa UDPCClient [2]

```
//receive a confirmation
buffer = new byte[1024];
packet = new DatagramPacket(buffer, buffer.length);
socket.receive(packet); //RECEIVE
String confirmation = new String(packet.getData(), 0, packet.getLength());
System.out.println("Received: " + confirmation);

Thread.sleep(3000);
}

} catch (IOException | InterruptedException ex) {
    ex.printStackTrace();
} //CLOSE
}
```

Klasa UDPServer [1]

```
public class UDPServer {  
    public static void main(String[] args) {  
        try ( DatagramSocket socket = new DatagramSocket(11111)) { //SOCKET -> BIND  
            while (true) {  
                //receive a reading  
                byte[] buffer = new byte[1024];  
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);  
                socket.receive(packet); //RECEIVE  
                String json = new String(packet.getData(), 0, packet.getLength());  
                Reading reading = Reading.fromJson(json);  
                System.out.println("Received: " + reading);  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Klasa UDPServer [2]

```
//send a confirmation
String confirmation = "OK";
buffer = confirmation.getBytes();
packet = new DatagramPacket(buffer, buffer.length,
packet.getAddress(), packet.getPort());
socket.send(packet); //SEND
System.out.println("Sent: " + confirmation);
}
} catch (IOException ex) {
ex.printStackTrace();
} //CLOSE
}
```

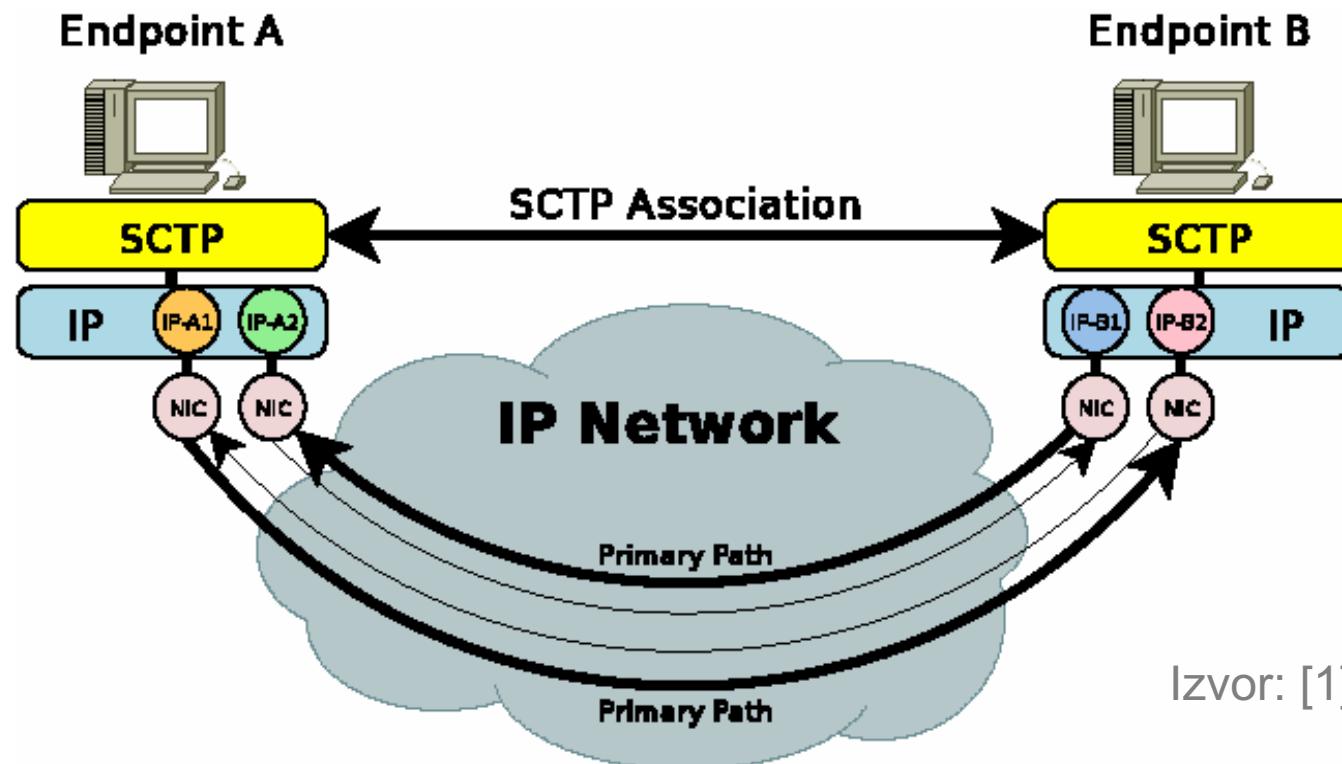


Protokol SCTP

Stream Control Transport Protocol (SCTP)

- Pouzdani protokol transportnog sloja
- Sličan je TCP-u po tome što je pouzdan
- Sličan je UDP-u po tome što se prijenos podataka odvija porukama, a ne korištenjem tokova okteta kao kod TCP-a
- *Multihoming*
 - Omogućava (mrežnu) redundanciju jer krajnja točka može biti predstavljena s više adresa za slanje i primanje podataka
 - Pri tome je port isti na svim redundantnim adresama
- Svaka asocijacija podržava više logičkih tokova poruka

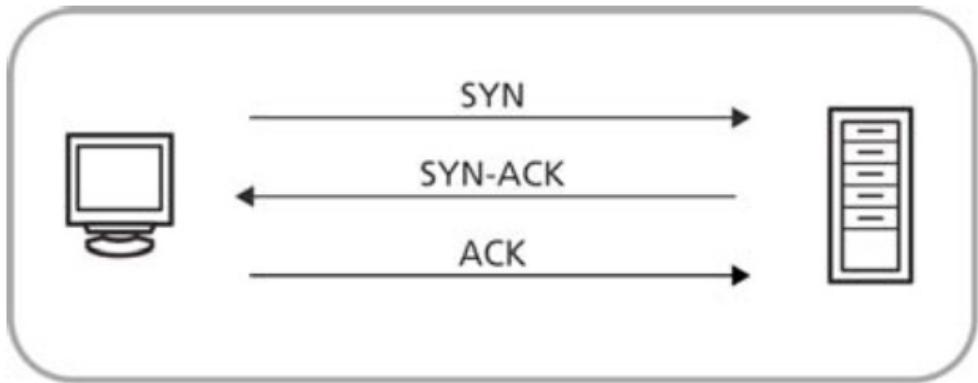
Multihoming kod SCTP-a



Izvor: [1]

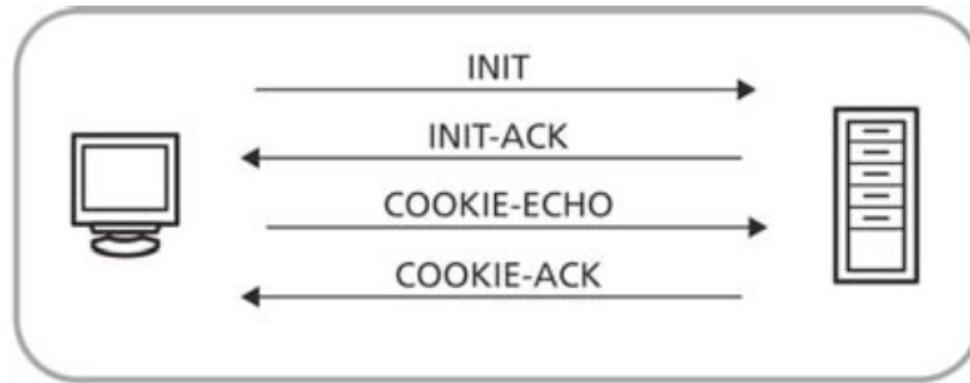
Uspostava i prekidanje konekcije kod TCP-a i SCTP-a

TCP: three-way handshake



TCP Connection Initiation

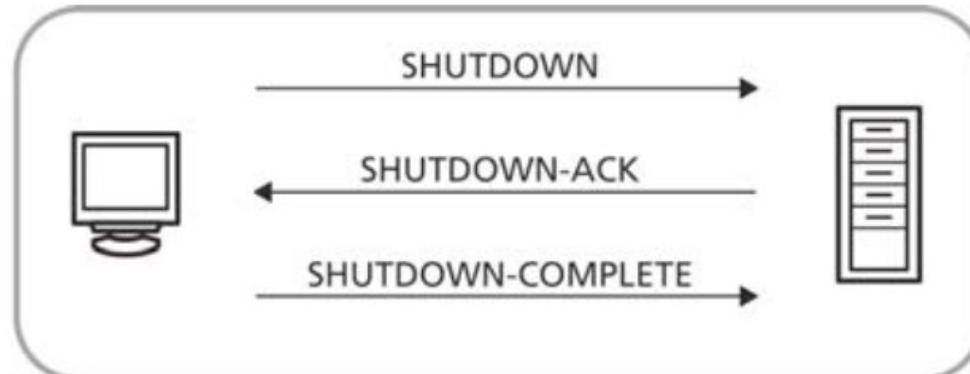
SCTP: four-way handshake



SCTP Connection Initiation



TCP Connection Termination
(Half-open shown by red)



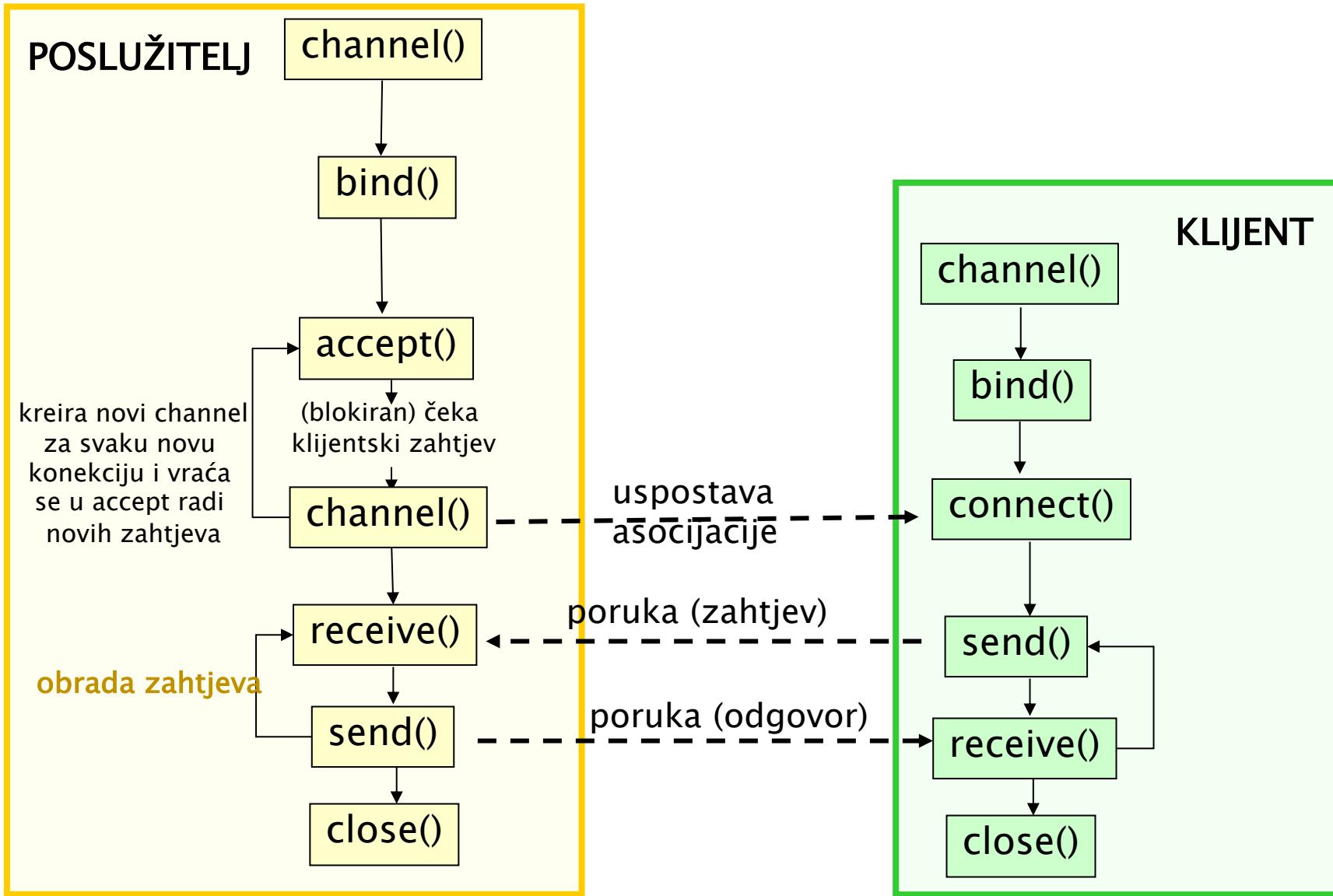
SCTP Close Connection

Izvor: [2]

Razlike između protokola TCP, UDP i SCTP

	TCP	UDP	SCTP
Pouzdanost	Da	Ne	Da
Konekcijska komunikacija	Da	Ne	Da
Prijenos podataka	Tok okteta	Poruke	Poruke
Kontrola toka	Da	Ne	Da
Kontrola zagušenja	Da	Ne	Da
Tolerancija na ispade	Ne	Ne	Da (<i>Multihoming</i>)
Redoslijed	Uređen	Neuređen	Djelomično uređen
Sigurnost	Da	Da	Poboljšana

Konekcijska komunikacija pomoću kanala SCTP



Obilježja protokola SCTP

- model klijent-poslužitelj
- vremenska ovisnost procesa
 - poslužitelj mora biti aktivan za primanje datagrama
- klijent mora znati identifikator poslužitelja
- tranzijentna komunikacija
- asinkrona komunikacija
 - klijent šalje datagram i nastavlja obradu, nema blokiranja pošiljatelja
- pouzdana komunikacija

SCTP: implementacija klijenta

1. Kreirati klijentski kanal:

```
SctpChannel clientSctpChannel =  
SctpChannel.open().bind(clientSocketAddress));
```

2. Povezati se s poslužiteljem:

```
clientSctpChannel.connect(serverSocketAddress, 1, 1);
```

3. Komunikacija porukama s poslužiteljem

```
clientSctpChannel.send(byteBuffer, messageInfo);  
clientSctpChannel.receive(byteBuffer, null, null);
```

4. Zatvoriti kanal:

```
clientSctpChannel.close();
```

SCTP: implementacija poslužitelja

1. Kreirati poslužiteljski kanal:

```
SctpServerChannel sctpServerChannel;  
sctpServerChannel = SctpServerChannel.open().bind(new  
InetSocketAddress(PORT));
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog kanala:

```
SctpChannel clientSctpChannel = sctpServerChannel.accept()
```

3. Komunikacija porukama s klijentom

```
clientSctpChannel.receive(byteBuffer, null, null);  
clientSctpChannel.send(byteBuffer, messageInfo);
```

4. Zatvoriti kopiju kanala:

```
clientSctpChannel.close();
```

5. Zatvoriti poslužiteljski kanal:

```
sctpServerChannel.close();
```

Primjer SCTP klijenta i poslužitelja

- Klijent šalje senzorska očitanja u JSON formatu
- Server parsira primljena senzorska očitanja i ispisuje ih na konzolu

Klasa SCTPClient [1]

```
public class SCTPClient {  
  
    public static void main(String[] args) {  
        InetSocketAddress serverSocketAddress = new InetSocketAddress("localhost", 54321);  
        InetSocketAddress clientSocketAddress = new InetSocketAddress(54325);  
  
        try ( SctpChannel clientSctpChannel = SctpChannel.open().bind(clientSocketAddress)) { //CHANNEL -> BIND  
            clientSctpChannel.connect(serverSocketAddress, 1, 1); //CONNECT  
  
            while (true) {  
                ...  
            }  
        } catch (IOException | InterruptedException ex) {  
            ex.printStackTrace();  
        } //CLOSE  
    }  
}
```

Klasa SCTPCClient [1]

```
//start sending readings
while (true) {
    //send a reading
    Reading reading = new Reading("id_4", "temperature", (Math.random() * 60) - 20, "C");
    byte[] message = reading.toJson().getBytes("UTF-8");
    ByteBuffer byteBuffer = ByteBuffer.wrap(message);
    System.out.println("Sent: " + reading);
    MessageInfo messageInfo = MessageInfo.createOutgoing(null, 0);
    clientSctpChannel.send(byteBuffer, messageInfo); //SEND

    //receive a confirmation
    byteBuffer = ByteBuffer.allocate(1024);
    messageInfo = clientSctpChannel.receive(byteBuffer, null, null); //RECEIVE
    message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());
    String confirmation = new String(message, "UTF-8");
    System.out.println("Received: " + confirmation);

    Thread.sleep(2000);
}
```

Klasa SingleClientSCTPServer [1]

```
public class SingleClientSCTPServer {  
  
    private int port;  
  
    public SingleClientSCTPServer(int port) {  
        this.port = port;  
    }  
  
    public static void main(String[] args) {  
        SingleClientSCTPServer server = new SingleClientSCTPServer(12345);  
        server.start();  
    }  
}
```

Klasa SingleClientSCTPServer [2]

```
public void start() {  
    SocketAddress serverSocketAddress = new InetSocketAddress(54321);  
  
    try ( SctpServerChannel stcpServerChannel = SctpServerChannel.open().bind(serverSocketAddress) ) { //CHANNEL -> BIND  
        SctpChannel clientSctpChannel = stcpServerChannel.accept(); //ACCEPT -> CHANNEL  
  
        while (true) {  
            try {  
                //receive a reading  
                ByteBuffer byteBuffer = ByteBuffer.allocate(1024);  
                MessageInfo messageInfo = clientSctpChannel.receive(byteBuffer, null, null); //RECEIVE  
                byte[] message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());  
                String jsonReading = new String(message, "UTF-8");  
                Reading reading = Reading.fromJson(jsonReading);  
                System.out.println("Received: " + reading);  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Klasa SingleClientSCTPServer [3]

```
//send a confirmation
String confirmation = "OK";
message = confirmation.getBytes("UTF-8");
byteBuffer = ByteBuffer.wrap(message);
System.out.println("Sent: " + confirmation);
messageInfo = MessageInfo.createOutgoing(null, 0);
clientSctpChannel.send(byteBuffer, messageInfo); //SEND
} catch (IOException ex) {
    ex.printStackTrace();
}
}
} catch (IOException ex) {
    ex.printStackTrace();
}
} //CLOSE
}
```

Klasa SCTPServer [1]

```
public class SCTPServer {  
    private int port;  
  
    public SCTPServer(int port) {  
        this.port = port;  
    }  
  
    public void start() {  
        SocketAddress serverSocketAddress = new InetSocketAddress(54321);  
  
        try ( SctpServerChannel stcpServerChannel = SctpServerChannel.open().bind(serverSocketAddress)) { //CHANNEL -> BIND  
            while (true) {  
                SctpChannel clientSctpChannel = stcpServerChannel.accept(); //ACCEPT -> CHANNEL  
                new Thread(new ServerTask(clientSctpChannel)).start();  
            }  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
  
    public static void main(String[] args) {  
        SCTPServer server = new SCTPServer(12345);  
        server.start();  
    }  
}
```

Klasa SCTPServer [2]

```
private class ServerTask implements Runnable {  
  
    private SctpChannel sctpChannel;  
  
    public ServerTask(SctpChannel sctpChannel) {  
        this.sctpChannel = sctpChannel;  
    }  
  
    @Override  
    public void run() {  
        while (true) {  
            try {  
                //receive a reading  
                ByteBuffer byteBuffer = ByteBuffer.allocate(1024);  
                MessageInfo messageInfo = sctpChannel.receive(byteBuffer, null, null); //RECEIVE  
                byte[] message = Arrays.copyOfRange(byteBuffer.array(), 0, messageInfo.bytes());  
                String jsonReading = new String(message, "UTF-8");  
                Reading reading = Reading.fromJson(jsonReading);  
                System.out.println("Received: " + reading);  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Klasa SCTPServer [2]

```
//send a confirmation
String confirmation = "OK";
message = confirmation.getBytes("UTF-8");
byteBuffer = ByteBuffer.wrap(message);
System.out.println("Sent: " + confirmation);
messageInfo = MessageInfo.createOutgoing(null, 0);
sctpChannel.send(byteBuffer, messageInfo); //SEND
} catch (IOException ex) {
    ex.printStackTrace();
}
}
}
}
```

Literatura

- [1] Dreibholz, T., Zhou, X., Becke, M., Pulinthanath, J., Rathgeb, E. and Du, W., 2010. On the security of reliable server pooling systems. International Journal of Intelligent Information and Database Systems, 4(6), pp.552-578.
- [2] [Introduction to the Stream Control Transmission Protocol \(SCTP\): The next generation of the Transmission Control Protocol \(TCP\)](#)

Programska potpora komunikacijskim sistavima

11. predavanje, 31. svibnja 2023.
doc. dr. sc. Josip Vuković

SQL i baze podataka



- Predavanje izradio izv. prof. dr. sc. Marin Vuković

Creative Commons



- **slobodno smijete:**

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **remiksirati** — 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 licencem koja je ista ili slična ovoj.

U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela. Najbolji način da to učinite je linkom na ovu internetsku stranicu.

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 licencije preuzet je s <http://creativecommons.org/>.

Sadržaj predavanja

- Dizajn baze podataka
- Kreiranje korisnika
- SQL upiti
- Primjeri iz koda

Kreiranje baze podataka

- Radimo web aplikaciju u kojoj želimo imati:
 - Studente
 - Predmete
 - Ocjene studenata na predmetima
- Dizajn baze i organizacija podataka
 - Važan korak koji će nam omogućiti jednostavno rukovanje podacima i eventualna **proširenja**
 - Također, razmisliti i o indeksiranju baze -> brža pretraga, no koje podatke indeksirati?

Kreiranje baze podataka - tablice

- Radimo web aplikaciju u kojoj želimo imati:
 - Studente -> tablica studenti
 - Predmete → tablica predmeti
 - Ocjene studenata na predmetima -> tablica ocjene

Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default	Extra	Encoding	Collation
id	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_increment	<input type="checkbox"/>	<input type="checkbox"/>
ime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	None	<input type="checkbox"/>	UTF-8 Unicode	<input type="checkbox"/> utf8_general_ci <input type="checkbox"/>
prezime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	None	<input type="checkbox"/>	UTF-8 Unicode	<input type="checkbox"/> utf8_general_ci <input type="checkbox"/>

- Ime i prezime?
 - hrvatski znakovi!
 - koje kodiranje koristiti (*Encoding / Collation*)?

Kreiranje baze podataka – tipovi podataka

Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default	Extra	Encoding	Collation
id	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_increment	<input type="checkbox"/>	<input type="checkbox"/>
ime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	UTF-8 Unicode
prezime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	UTF-8 Unicode

- Tipovi podataka
 - INT – integer definirane duljine (11)
 - Dobar za pretraživanje!
 - VARCHAR – promjenjivi niz znakova definirane duljine (30)
 - Relativno loš za pretraživanje!
- Još tipova podataka za različite svrhe...

Kreiranje baze podataka – tipovi podataka 2

- Još neki tipovi podataka...

Brojevi

TINYINT
SMALLINT
MEDIUMINT
INT
BIGINT
FLOAT
DOUBLE
DOUBLE PRECISION
REAL
DECIMAL

Znakovi, string, datoteke...

CHAR
VARCHAR
TINYTEXT
TEXT
MEDIUMTEXT
LONGTEXT
TINYBLOB
MEDIUMBLOB
BLOB
LONGBLOB

Vrijeme i datum

DATE
DATETIME
TIMESTAMP
TIME
YEAR

Kreiranje baze podataka - postavke

Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default	Extra	Encoding	Collation
id	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI		auto_increment	<input type="checkbox"/>	<input type="checkbox"/>
ime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	UTF-8 Unicode
prezime	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		NULL	None	<input type="checkbox"/>	UTF-8 Unicode

- Allow Null
 - Smije li podatak biti null – baca grešku!
- Extra
 - Auto_increment
 - kod dodavanja novog "retka" automatski se povećava za jedan
 - Brisanje?
 - On update CURRENT_TIMESTAMP
 - Serial default value
 - Npr. 0 za integer
- Key
 - Važno – koji je primarni ključ?
 - Povezivanje više tablica s ključevima

Kreiranje baze podataka - konačno

studenti

Field	Type	Length
id	INT	11
ime	VARCHAR	30
prezime	VARCHAR	30

ocjene

Field	Type	Length
id	INT	11
predmetID	INT	11
studentID	INT	11
ocjena	INT	11

predmeti

Field	Type	Length
id	INT	11
naziv	VARCHAR	60

Dodavanje novog korisnika nad bazom

- Imamo bazu – moramo kreirati korisnika koji će joj pristupati
 - Zapravo, taj korisnik će predstavljati našu web aplikaciju
 - Točnije, web aplikacija će imati sve ovlasti kao i korisnik baze

```
grant all privileges on PPKS.* to ppksuser@localhost identified by  
'ppksjakipassword';
```

- Je li uvijek potrebno navesti „all”?
 - Read, write...

SQL naredbe

- Naredbe za manipuliranje podacima u bazi podataka
- *Structured Query Language*
- Vrlo slične za različite baze podataka
- Najčešće:
 - **INSERT** – dodavanje
 - **UPDATE** – ažuriranje
 - **SELECT** – čitanje
 - **DELETE** – brisanje
- Ali i još puno puno drugih...
 - **CREATE DATABASE**
 - **ALTER DATABASE**
 - **CREATE TABLE**
 - **ALTER TABLE**
 - ...

SQL naredbe - dodavanje

id	ime	prezime
1	Marin	Vuković
2	Krešimir	Pripužić



id	ime	prezime
1	Marin	Vuković
2	Krešimir	Pripužić
3	Test-ime	Test-prezime

```
INSERT INTO studenti (ime, prezime) VALUES ('Test-ime', 'Test-prezime');
```

(uočiti: id je “auto-increment”)

SQL naredbe - dodavanje

id	predmetID	studentID	ocjena
1	1	1	2



id	predmetID	studentID	ocjena
1	1	1	2
2	1	2	3

```
INSERT INTO ocjene (predmetID, studentID, ocjena) VALUES (1, 2, 3)
```

Što ako više puta pozovemo istu naredbu?

id	predmetID	studentID	ocjena
1	1	1	2
2	1	2	3
3	1	2	3
4	1	2	3

“Idempotentnost”

SQL naredbe - ažuriranje

Ne dodaje redak nego mijenja postojeći

id	predmetID	studentID	ocjena
5	1	2	3



id	predmetID	studentID	ocjena
5	1	2	5

Uočiti: id=5 - zašto?

```
UPDATE ocjene SET predmetID=1, ocjena=5 WHERE studentID=2
```

Što ako ne postoji takav "redak"?

```
UPDATE ocjene SET predmetID=1, ocjena=5 WHERE studentID=1
```

SQL naredbe – čitanje/odabir

Dohvaća podatke iz baze podataka

```
SELECT * FROM studenti
```

id	ime	prezime
1	Marin	Vuković
2	Krešimir	Pripužić
4	Test-ime	Test-prezime

```
SELECT * FROM studenti WHERE id=1
```

id	ime	prezime
1	Marin	Vuković

SQL naredbe – čitanje/odabir

Različite mogućnosti kod uvjeta (WHERE)

```
SELECT * FROM studenti WHERE ime="Marin"
```

id	ime	prezime
1	Marin	Vuković

```
SELECT * FROM studenti WHERE prezime LIKE '%ić%'
```

id	ime	prezime
1	Marin	Vuković
2	Krešimir	Pripužić

Poredavanje rezultata (sort)

```
SELECT * FROM studenti WHERE prezime LIKE '%ić%' ORDER BY prezime ASC;
```

id	ime	prezime
2	Krešimir	Pripužić
1	Marin	Vuković

SQL naredbe – čitanje/odabir

Sortiranje rezultata

```
SELECT * FROM studenti WHERE prezime LIKE '%ić%' ORDER BY prezime ASC;
```

id	ime	prezime
2	Krešimir	Pripužić
1	Marin	Vuković

Što ako ne postoje rezultati?

```
SELECT * FROM studenti WHERE prezime LIKE '%ne-postoji%' ORDER BY prezime ASC;
```

SQL naredbe – čitanje/odabir

Povezivanje podataka iz više tablica - UNION

```
SELECT prezime FROM studenti WHERE prezime LIKE '%ić%' UNION SELECT  
predmetID FROM ocjene WHERE ocjena=5;
```

prezime
Vuković
Pripužić
2

Složeni(ji) upiti

```
SELECT * FROM studenti WHERE id IN (SELECT studentID FROM ocjene WHERE ocjena = 5;
```

id	ime	prezime
2	Krešimir	Pripužić

SQL naredbe – brisanje

Brišemo cijelu tablicu

```
DELETE * FROM ocjene;
```

Brisanje s uvjetom

```
DELETE FROM ocjene WHERE ocjena=5;
```

Primjeri koda - konfiguracija

- Primjer PHP ali više-manje je slično u svim programskim jezicima
- Konfiguracija – postavke za spajanje na bazu podataka

```
$db host = "localhost"  
$db name = "PPKS"  
$username = "ppksuser"  
$password = "ppksjakipassword";
```

- Sjetimo se:

```
grant all privileges on PPKS.* to ppksuser@localhost identified by 'ppksjakipassword';
```

- Ovo smo unosili na MySQL konzoli ili u nekom alatu za pristup bazi
- Npr. konzola: mysql -u root -p

Primjeri koda – spajanje na bazu

- Spajanje na bazu podataka – klasa dbConnect:

```
<?php

> class dbConnect{
    private $_mysqli;

>     public function __construct(){
        include ('configuration.php');
        $this->mysqli = new mysqli($db_host, $username, $password, $db_name);
        $this->mysqli->select_db($db_name);
        if (mysqli_connect_errno()) {
            printf("Connect failed: %s\n", mysqli_connect_error());
            exit();
        }
    }
}
```

- I metode za različite manipulacije baze
 - Dodaj, ažuriraj, odaberi....

Primjeri koda - metode

```
function getStudents(){
    $query = "SELECT * FROM studenti";
    echo $query;
    $result = $this->mysqli->query($query) or die($this->mysqli->error.__LINE__);
    $results;
    $cnt=0;
    while ($row = $result->fetch_row()) {
        $cnt++;
        $results[$cnt] = $row;
    }
    return $results;
}

function getStudents2(){
    $query = "SELECT id, ime, prezime FROM studenti";
    echo $query;
    $result = $this->mysqli->query($query) or die($this->mysqli->error.__LINE__);
    $results;
    $cnt=0;
    while ($row = $result->fetch_row()) {
        $cnt++;
        $results[$cnt]["id"] = $row[0];
        $results[$cnt]["ime"] = $row[1];
        $results[$cnt]["prezime"] = $row[2];
    }
    return $results;
}
```

Primjeri koda - metode

```
function updateOcjene($studentID, $predmetID, $ocjena){
    $query = "UPDATE ocjene SET predmetID=".$predmetID.", ocjena=".$ocjena." WHERE studentID=".$studentID;
    echo $query;
    $this->mysqli->query($query) or die($this->mysqli->error.__LINE__);
}

function dodavanjeOcjene($studentID, $predmetID, $ocjena){
    $query = "INSERT INTO ocjene (predmetID, studentID, ocjena) VALUES (".$predmetID.", ".$studentID.", ".$ocjena.");
    echo $query;
    $this->mysqli->query($query) or die($this->mysqli->error.__LINE__);
}
function brisanjeOcjene($studentID, $predmetID){
    $query = "DELETE FROM ocjene WHERE studentID=".$studentID." AND predmetID=".$predmetID;
    echo $query;
    $this->mysqli->query($query) or die($this->mysqli->error.__LINE__);
}
```

Primjeri koda – poziv metoda

- `session_start()`
 - dajemo uputu poslužitelju weba da stvori sjednicu
 - praćenje sjednice, stanja, varijabli...
- `require_once(...)`
 - Učitaj datoteku jednom
- `$dbConnector`
 - Instanca klase dbConnect
 - Pozivamo metode klase

```
session_start();
require_once('dbconnect.php');
$dbConnector = new dbConnect();
$dbConnector->dodavanjeOcjene(1, 1, 2);
$dbConnector->updateOcjene(1, 1, 5);
$dbConnector->brisanjeOcjene(1, 1);
echo $dbConnector->getStudents();
```

Neki problemi ovakvog pristupa 1

- Vjerovanje korisniku?

```
function getStudents($ime){  
    $query = "SELECT * FROM studenti WHERE ime='".$ime."'";
```

- SQL injection!

```
SELECT * FROM studenti WHERE ime='Marin' OR '1'='1';
```

- Nikada ne smijemo vjerovati korisniku i njegov unos izravno slati na bazu
- Koristiti barem:

```
private function sanitize($input) {  
    if (get_magic_quotes_gpc()) {  
        $input = stripslashes($input);  
    }  
    $output = mysqli_real_escape_string($this->mysqli, $input);  
    return $output;  
}
```

Neki problemi ovakvog pristupa 2

- Danas se SQL upiti iz koda ne izvode ovako!
 - Iako su same SQL naredbe iste
- Koriste se „pripremljene izjave” (*prepared statements*)

```
$stmt = $pdo->prepare("SELECT * FROM studenti WHERE id=:id");
$stmt->execute(['id' => $id]);
$student = $stmt->fetch();
```

- Još bolje:
 - Perzistencija objekata u bazu
 - Npr. objekt „student”
 - Parametri ime, prezime
 - Čitav objekt se pohranjuje u i dohvata iz baze

Programska potpora komunikacijskim sustavima

Dvanaesto predavanje, 7. lipnja 2023.

Izv. prof. dr. sc. Krešimir Pripužić

Doc. dr. sc. Josip Vuković

Objektno relacijsko mapiranje (ORM)



Sadržaj predavanja

- Uvod
- Objektno-relacijsko mapiranje - ORM
- Java Persistence API - JPA
- Hibernate
- Primjeri korištenja
- Sažetak

Uvod

- Aplikacije koriste podatke
 - Potrebno brzo i efikasno spremanje i korištenje podataka
- Perzistencija podataka
- Što odabrati?
 - Tekstualne datoteke
 - SQL
 - NoSQL
 - Serijalizirani objekti

Objektno-relacijsko mapiranje (ORM)

- Povezuje dva (logički) „nekompatibilna” sustava
- Stvara se sloj između relacijskih baza podataka (Oracle, MySQL, H2, MongoDB, PostgreSQL, ...) i objektno orijentiranih programskih jezika (Java, C#, Python, ...)
- Omogućuje programeru „automatiziran” prijenos podataka iz objekata u bazu i obrnuto

JDBC (*Java Database Connectivity*)

- Javni API za spajanje na bazu podataka
- Svaka baza ima svoj API
- Problem u različitoj sintaksi SQL-a ovisno o bazi podataka
- Programer mora sam voditi računa o osnovnim CRUD (*create, read, update, delete*) operacijama nad bazom podataka

Dodavanje potrebne ovisnosti u pom.xml

```
<!--  
https://mvnrepository.com/artifact/com.h2database/h2  
-->  
  
<dependency>  
    <groupId>com.h2database</groupId>  
    <artifactId>h2</artifactId>  
    <version>2.1.210</version>  
</dependency>
```

Primjer korištenja JDBC-a [1]

```
public class JdbcMain {  
    static final String DB_URL = "jdbc:h2:~/students";  
    //static final String DB_URL = "jdbc:mysql://localhost/students";  
    static final String USER = "sa";  
    static final String PASS = "passwd";  
  
    public static void main(String[] args) {  
        try ( Connection connection = DriverManager.getConnection(DB_URL, USER, PASS); Statement statement =  
connection.createStatement()) {  
            //create a table  
            String sql = "CREATE TABLE Student " + "(id INT PRIMARY KEY, first_name VARCHAR(255), last_name  
VARCHAR(255));";  
            //String sql = "CREATE TABLE Student " + "(id INTEGER not NULL, " + " first_name VARCHAR(255), " + "  
last_name VARCHAR(255)" + " PRIMARY KEY ( id ));"  
            statement.executeUpdate(sql);  
  
            //insert a student  
            sql = "INSERT INTO Student VALUES (1, 'Ante', 'Antić');"  
            //sql = "INSERT INTO(id, first_name, last_name) Student VALUES (1, 'Ante', 'Antić')";  
            statement.executeUpdate(sql);  
  
            //print all students  
            printAllStudents(statement);  
    }  
}
```

Primjer korištenja JDBC-a [1]

```
//delete a student
sql = "DELETE FROM Student WHERE ID=1";
statement.executeUpdate(sql);

//print all students
printAllStudents(statement);

} catch (SQLException e) {
    e.printStackTrace();
}

}

private static void printAllStudents(final Statement statement) throws SQLException {
    String sql = "SELECT * FROM Student";
    ResultSet resultSet = statement.executeQuery(sql);
    while (resultSet.next()) {
        for (int i = 1; i <= resultSet.getMetaData().getColumnCount(); i++) {
            System.out.print(resultSet.getString(i) + " ");
        }
        System.out.println("");
    }
}
```

JPA (Java Persistence API)

- Specifikacija koja opisuje objektno-relacijsko mapiranje
- Postoji više implementacija koje podržavaju JPA
 - Hibernate
 - EclipseLink
 - Apache Open JPA
 - TopLink
 - ...
- Programer se bavi samo objektima, a ne mora SQL-om za rad s bazom podataka

JPA (Java Persistence API)

- Mapiranje podataka između baze i objektno orijentiranog jezika koristeći informacije sadržane u *metapodatcima*
 - *Metapodatci* mogu biti opisani u konfiguracijskom XML-u ili koristeći anotacije u samom programskom kodu ili kombinacijom anotacija i XML-a
 - XML konfiguracija uvijek nadjačava anotacije
- Koristi se jezik sličan SQL-u za slanje „direktnih“ upita u bazu (iz programskog koda) prilikom npr. testiranja

Hibernate

- Nije potrebno nasljeđivanje apstraktnih klasa/implementiranje sučelja
- POJO objekti „ne znaju” da će biti spremljeni u bazu podataka
- Relacije među klasama koje su podržane su:
 - *one-to-one*
 - *one-to-many*
 - *many-to-many*
- Podržana je refleksivnost (*one-to-many* relacija s objektima istog tipa kao i objekt nad kojim se provodi relacija)
- Za vrijeme *runtime-a* stvaraju se SQL upiti na temelju anotacija/XML-a

Dodavanje dodatne ovisnosti u pom.xml

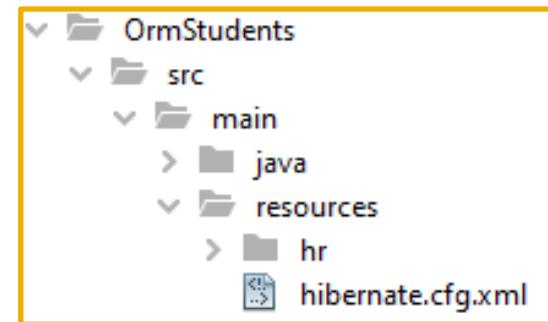
```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.0.2.Final</version>
    <type>pom</type>
</dependency>
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.210</version>
</dependency>
```

Klasa HibernateUtil

```
public class HibernateUtil {  
    private static SessionFactory factory = buildSessionFactory();  
  
    private static SessionFactory buildSessionFactory() {  
        try {  
            if (factory == null) {  
                StandardServiceRegistry registry = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();  
                Metadata metadata = new MetadataSources(registry).getMetadataBuilder().build();  
                factory = metadata.getSessionFactoryBuilder().build();  
            }  
            return factory;  
        } catch (Throwable ex) {  
            throw new ExceptionInInitializerError(ex);  
        }  
    }  
  
    public static SessionFactory getSessionFactory() {  
        return factory;  
    }  
  
    public static void shutdown() {  
        getSessionFactory().close();  
    }  
}
```

Konfiguracijska datoteka hibernate.cfg.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:~/students</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password">passwd</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="hbm2ddl.auto">create</property>
        <mapping resource="hr/fer/zkist/ppks/students/student.hbm.xml"/>
    <!--      <mapping class="hr.fer.zkist.ppks.students.Student"/>-->
    </session-factory>
</hibernate-configuration>
```



Klasa Student [1]

```
public class Student {  
    private String firstName;  
    private String lastName;  
    private String jmbag;  
  
    public Student(String firstName, String lastName, String jmbag) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.jmbag = jmbag;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
}
```

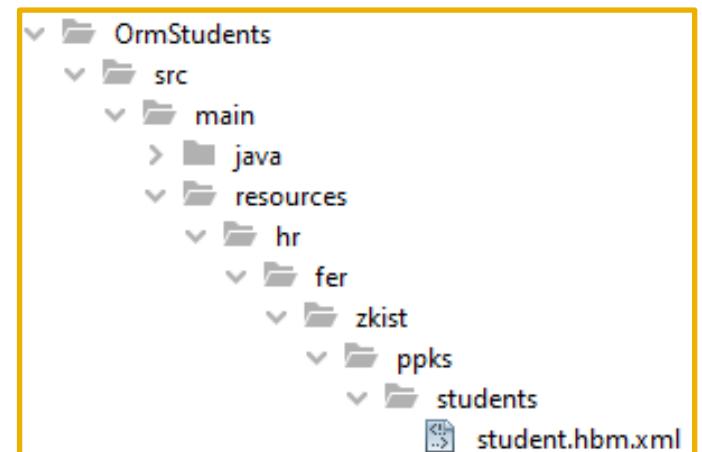
Klasa Student [2]

```
public String getLastName() {  
    return lastName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}  
  
public String getJmbag() {  
    return jmbag;  
}  
  
public void setJmbag(String jmbag) {  
    this.jmbag = jmbag;  
}  
  
@Override  
public String toString() {  
    return "Student{" + "firstName=" + firstName + ", lastName=" + lastName + ", jmbag=" + jmbag + '}';  
}
```

Datoteka Student.hbm.xml

```
<hibernate-mapping>
    <class name="hr.fer.zkist.ppks.students.Student" table="Student">
        <id name="jmbag" column="id">
            <generator class="assigned" />
        </id>
        <property name="firstName" column="first_name" />
        <property name="lastName" column="last_name" />
    </class>
</hibernate-mapping>
```

Student		
id	first_name	last_name



Primjer korištenja – klasa OrmMain

```
Session session = HibernateUtil.getSessionFactory().openSession();

//add a student
session.beginTransaction();
Student student = new Student("Ante", "Antić", "1");
session.persist(student);
session.getTransaction().commit();

//print all students
List<Student> list = session.createQuery("FROM Student", Student.class).list();
System.out.println(list);

//delete a student
System.out.println("deleting");
session.beginTransaction();
session.remove(session.get(Student.class, "1"));
session.getTransaction().commit();

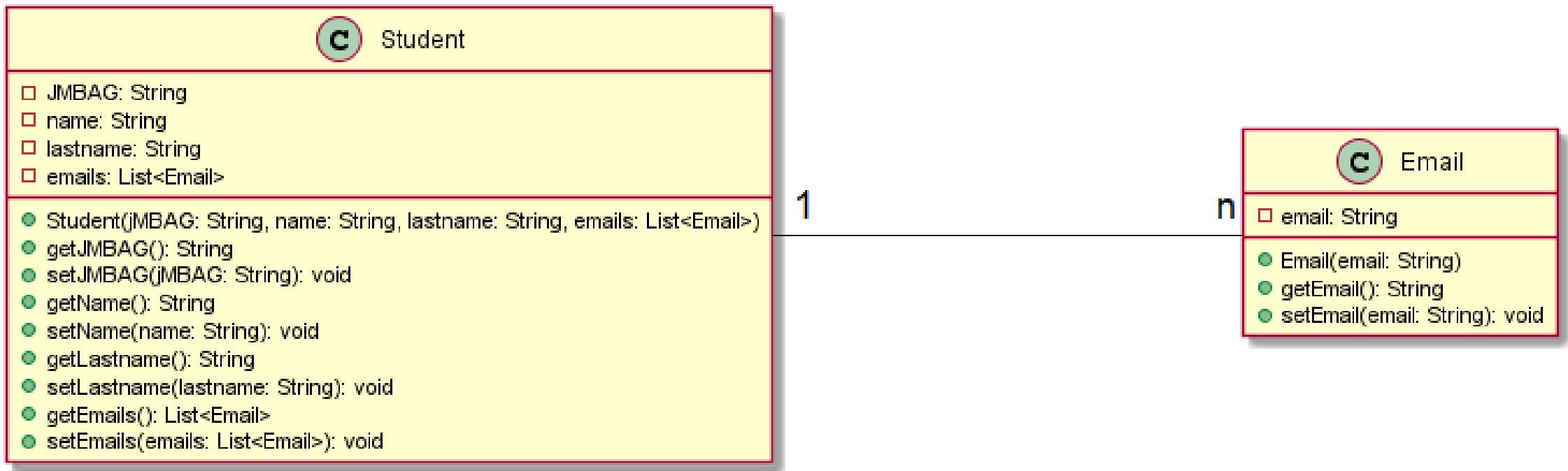
//print all students
list = session.createQuery("FROM Student", Student.class).list();
System.out.println(list);

HibernateUtil.shutdown();
```

Klasa Student s anotacijama (umjesto Student.hbm.xml)

```
@Entity  
@Table(name = "Student")  
public class Student {  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Column(name = "last_name")  
    private String lastName;  
  
    @Id  
    @Column(name = "Id", unique = true, nullable = false)  
    private String jmbag;  
  
    ...  
}
```

UML dijagram klasa primjera *one-to-many*



Klasa Student

```
@Entity
@Table(name = "Student")
public class Student {
    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Id
    @Column(name = "id", unique = true, nullable = false)
    private String jmbag;

    @OneToMany(cascade=CascadeType.PERSIST)
    @JoinColumn(name="student_id")
    private List<Email> emails;

    public Student(String firstName, String lastName, String jmbag, List<Email> emails) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.jmbag = jmbag;
        this.emails = emails;
    }
}
```

Klasa Email

```
@Entity
@Table(name = "Email")
public class Email {

    @Id
    @Column(name = "email")
    private String email;

    public Email(String email) {
        this.email = email;
    }

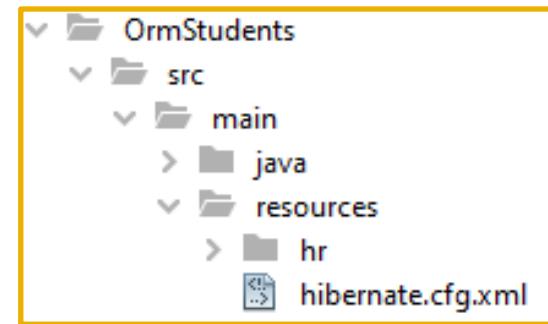
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Email{" + "email=" + email + '}';
    }
}
```

Konfiguracijska datoteka hibernate.cfg.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:~/students</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password">passwd</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="hbm2ddl.auto">create</property>
        <!--<mapping class="hr.fer.zkist.ppks.students.Student"/><!--&gt;
        &lt;mapping class="hr.fer.zkist.ppks.orm.one_to_n.Student"/&gt;
        &lt;mapping class="hr.fer.zkist.ppks.orm.one_to_n.Email"/&gt;
    &lt;/session-factory&gt;
&lt;/hibernate-configuration&gt;</pre>
```



Primjer korištenja – klasa OrmEmailsMain

```
public class OrmEmailsMain {  
  
    public static void main(String[] args) {  
        Session session = HibernateUtil.getSessionFactory().openSession();  
  
        //add a student  
        session.beginTransaction();  
  
        Email e1 = new Email("ivo.ivic@gmail.com");  
        Email e2 = new Email("ivo.ivic@yahoo.com");  
  
        List<Email> emails = new ArrayList<>();  
        emails.add(e1);  
        emails.add(e2);  
  
        Student student = new Student("Ante", "Antić", "1", emails);  
        session.persist(student);  
        session.getTransaction().commit();  
  
        ...  
    }  
}
```

Sadržaj baze podataka

Email	
email	student_id
ivo.ivic@gmail.com	1
ivo.ivic@yahoo.com	1

Student		
id	first_name	last_name
1	Ivo	Ivic

Sažetak

- Danas nema aplikacije bez podataka
- Aplikacijski kod bi trebao biti „neovisan” o korištenim platformama
- JPA + Hibernate je koristan alat koji olakšava programerima pohranu i korištenje podataka iz SQL baza podataka bez pisanja SQL naredbi u kodu
- Automatizirana manipulacija podatcima u „pozadini” tijekom izvođenja programa
- Jednostavna zamjena baza podataka kroz konfiguracijsku datoteku bez ponovnog prevodenja samog programskog koda



Spring Boot & Hibernate

Primjer RESTful web-usluge

resurs	podržane metode	šalje	svrha
/persons	GET	osoba	vraća listu osoba
	POST		stvara novu osobu
/persons/{id}	GET	osoba	vraća osobu s ID-om
	DELETE		briše osobu
	PUT	osoba	mijenja podatke o osobi

Pohrana resursa u REST usluzi

- Originalna usluga je pohranjivala osobe u memoriji u listi
- Poboljšana aplikacija će koristiti Hibernate za pohranu osoba u bazi podataka H2

Klasa Person s anotacijama

```
@Entity  
@Table(name = "Person")  
@JsonIgnoreProperties({"hibernateLazyInitializer","handler"})  
public class Person {  
  
    @Id  
    @Column(name = "id", unique = true, nullable = false)  
    private final long id;  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Column(name = "last_name")  
    private String lastName;  
  
    @Column(name = "phone")  
    private String phone;  
  
    @Column(name = "room")  
    private String room;  
  
    public Person() {  
        this.id = 0;  
    }  
}
```

...

Sučelje PersonRepository

```
@Repository  
public interface PersonRepository extends JpaRepository<Person, Long> {  
}
```

Klasa PersonService [1]

```
@Service
public class PersonService implements PersonInterface {

    @Autowired
    private PersonRepository personRepository;

    @Override
    public List<Person> get() {
        List<Person> persons = personRepository.findAll();
        Collections.sort(persons, Comparator.comparing(Person::getId));
        return persons;
    }

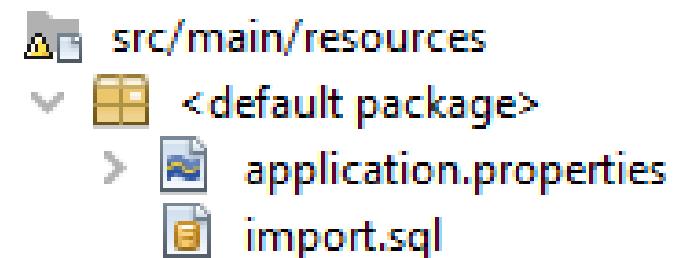
    @Override
    public void insert(Person person) {
        personRepository.save(person);
    }
}
```

Klasa PersonService [2]

```
@Override  
public Person get(long id) {  
    return personRepository.getById(id);  
}  
  
@Override  
public boolean delete(long id) {  
    boolean found = personRepository.existsById(id);  
    personRepository.deleteById(id);  
    return found;  
}  
  
@Override  
public boolean update(Person person, long id) {  
    boolean found = personRepository.existsById(id);  
    personRepository.save(person);  
    return found;  
}  
}
```

Dodavanje osoba u bazu podataka H2 pri pokretanju korištenjem datoteke import.sql

```
INSERT INTO Person values(0, 'Goran', 'Delač', '01/6129-549', 'D-339-2');
INSERT INTO Person values(1, 'Marin', 'Šilić', '01/6129-549', 'D-339-2');
INSERT INTO Person values(2, 'Marin', 'Vuković', '01/6129-658', 'C07-04');
INSERT INTO Person values(3, 'Krešimir', 'Pripužić', '01/6129-745', 'C08-17');
```



Dodavanje potrebnih ovisnosti u pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```