

Java-tečaj: Web front-end tehnologije



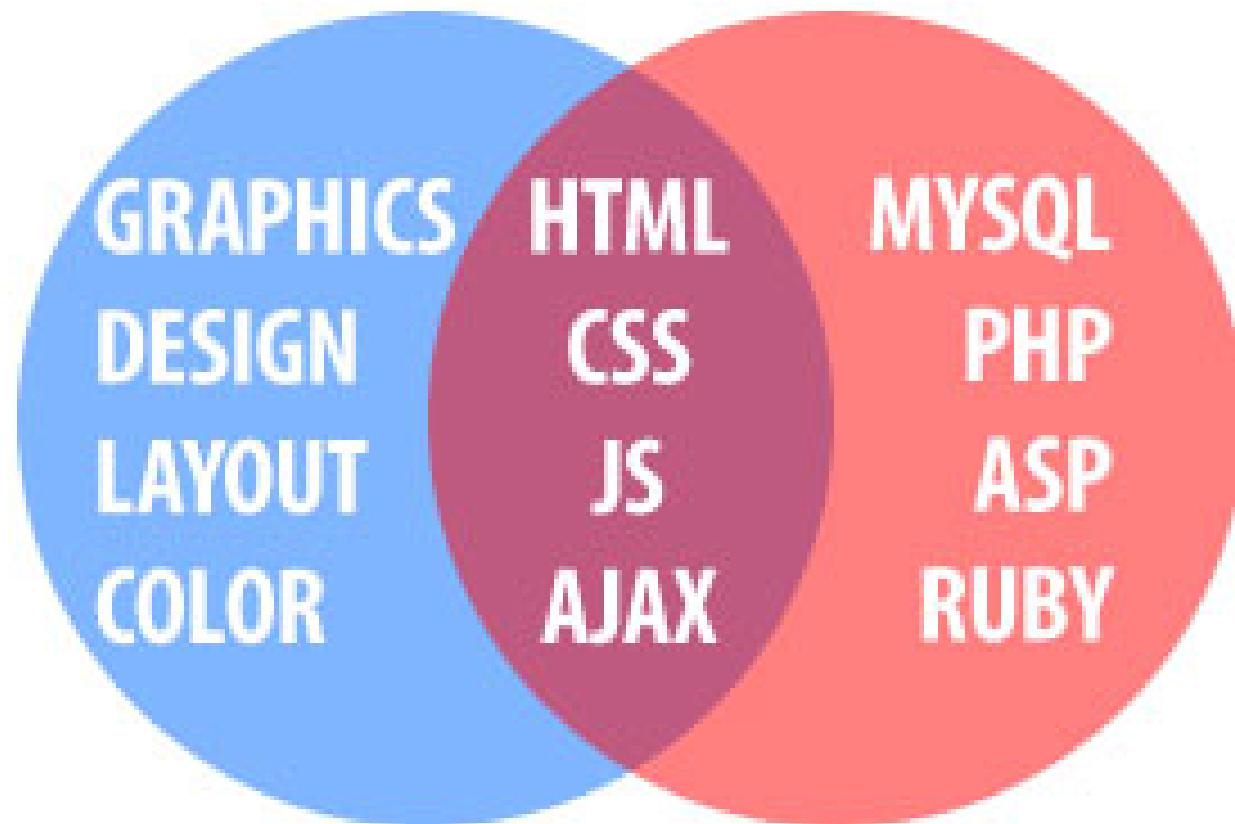
Marko Čupić

Zagreb, 2019.

Web front-end?

Front End design
code performance json
responsive Web semantic
svg slice canvas
mobile pages developer
facebook ajax jQuery
photoshop prototyping
javascript backbone.js UX / UI
wireframe SEO

Web front-end?



WEB DESIGN / WEB DEVELOPMENT



JavaScript

JavaScript

- JavaScript je objektno-orientirani programski jezik (s mnoštvom drugih podržanih paradigma)
 - Temeljen na prototipovima, ne klasičnim razredima
- Dinamički tipizirani jezik:
 - Variable i metode ne definiraju tipove podataka
 - Provjere se rade tijekom izvođenja programa
 - Variable mogu u različitim trenutcima čuvati podatke različitog tipa

JavaScript

- Podržava funkcije, uvjetna izvođenja i petlje, definiranje i stvaranje objekata, rad s poljima, stringovima, rad s timerima, ...

```
var broj = 17;  
var ime = "Pero";  
var datum = new Date();  
var polje1 = [1, 1, 2, 3, 5, 8, 13, 21];  
var polje2 = ["Ivana", "Pero", "Jasna", "Ante"];  
var zastavica = true;
```

```
var x; // samo deklarira varijablu; sadržaj je undefined  
if(x == undefined) {...} // je!  
x = 7;  
if(x == undefined) {...} // sada više nije!
```

JavaScript

- Doseg varijable definirane s `var` je čitava funkcija, neovisno o tome gdje je u funkciji definiran.
- Nove verzije JavaScripta uvode `let` čiji je doseg blok u kojem je definiran (petlja, tijelo od if-a, funkcija, ...)
- Preporuka: koristite `let!`

JavaScript

- Razlikuje strogu jednakost (==) i slabu jednakost (==)

```
console.log("Provjera 1: " + ("7"==undefined));    false
console.log("Provjera 2: " + ("7"==7));            true
console.log("Provjera 3: " + ("1"==true));          true
console.log("Provjera 4: " + ("0"==false));         true
console.log("Provjera 5: " + ("7"===7));           false
console.log("Provjera 6: " + ("7"==="7"));          true
console.log("Provjera 7: " + ("1"===true));         false
console.log("Provjera 8: " + ("0"===false));        false
console.log("Provjera 9: " + ("7"==="7"));          true
console.log("Provjera 10: " + (7===7));             true
```

JavaScript

- Tutorijala ima mnoštvo; preporuka:
<http://www.w3schools.com/js/default.asp>
- Stvoren u tvrtki Netscape 1995.
 - različita imena: Mocha → LiveScript → JavaScript
- Standardiziran kao ECMAScript kroz niz inačica
- Danas aktualna inačica ECMAScript 6

JavaScript

- Može se koristiti kao samostalan skriptni jezik
- Java platforma (JRE) ima ugrađene razrede koji omogućavaju da se iz Java poziva i izvršava skripta pisana u JavaScriptu
- Svaki web-preglednik također ima ugrađenu podršku za izvođenje skripti pisanih u JavaScriptu (a koje su dio HTML-dokumenta, bilo izravno bilo da ih se učitava izvana)

JavaScript

- Direktno umetanje skripte:

```
<script><!--  
 // ovdje idu naredbe skripte...  
//--></script>
```

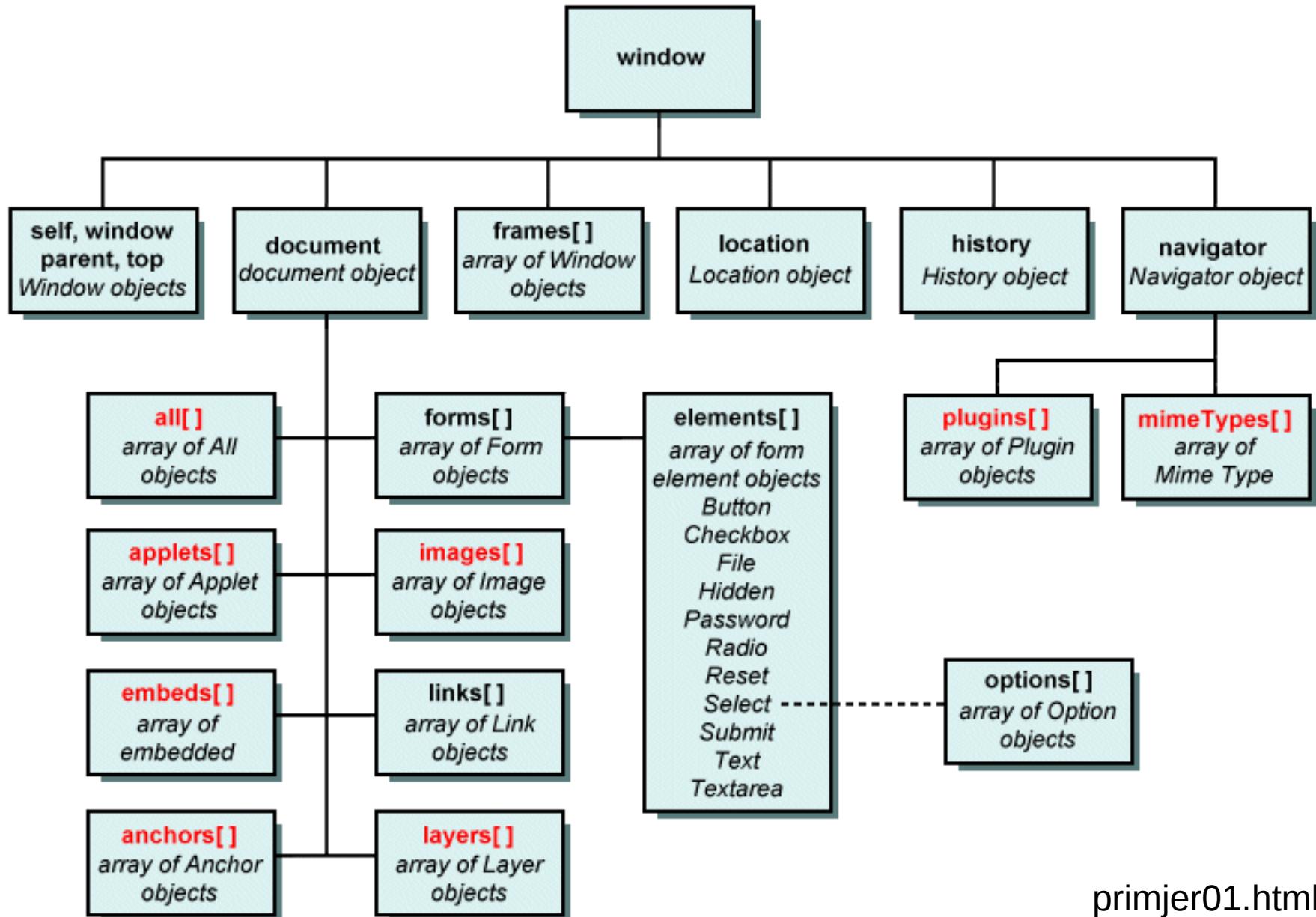
- Učitavanje skripte izvana:

```
<script src="js/htmlescaping.js"></script>
```

JavaScript

- Ako se JavaScript izvodi u web-pregledniku, tada na raspolaganju ima nekoliko objekata koji su veza prema pregledniku i trenutnom dokumentu
- **window** predstavlja prozor preglednika
- **window.document** predstavlja trenutni HTML dokument, tj. stablo tagova dobiveno parsiranjem
- **window.location** nudi informacije o adresi trenutnog HTML dokumenta (kao i mogućnost odlaska na novu adresu)

JavaScript



JavaScript funkcije

- Ključna riječ **function**
- Argumenti mogu i ne moraju biti imenovani
- Ne navodi se tip argumenata
- Funkcije su objekti (mogu se pamtitи u varijablama)

JavaScript funkcije

```
function zbroji(a,b) {  
    return a+b;  
}  
  
function zbroji2() {  
    let suma = 0;  
    for(let i = 0; i < arguments.length; i++) suma += arguments[i];  
    return suma;  
}  
  
var z = zbroji2;  
z(1,2,3,4,5);
```

JavaScript funkcije

```
function osoba(ime,prezime) { // koristimo kao konstruktorsku funkciju
    this.ime = ime;
    this.prezime = prezime;
}
// ne pozivati: var v = osoba(x, y);

let pero = new osoba("Pero", "Perić"); // new alocira objekt, on je this u funkciji

// alternativa: svaku funkciju možemo pozvati s call ili apply; prvi argument
// je ono na što u funkciji treba biti this
let pero = {};
osoba.call(pero, "Pero", "Perić");

let ivo = {};
osoba.apply(ivo, ["Ivo", "Ivić"]);

// općenito: this u funkciji pokazuje kome taj kod "pripada"; za globalne funkcije
// to će biti objekt window
```

Vidi: http://www.w3schools.com/js/js_function_definition.asp

JavaScript: varijable

- Ključna riječ **var** deklarira lokalnu varijablu dosega funkcije (osim ako nije u globalnom kontekstu kada je globalna)

```
var a = 7; // globalna varijabla
function f() {
    var b = 8; // lokalna varijabla
}
```

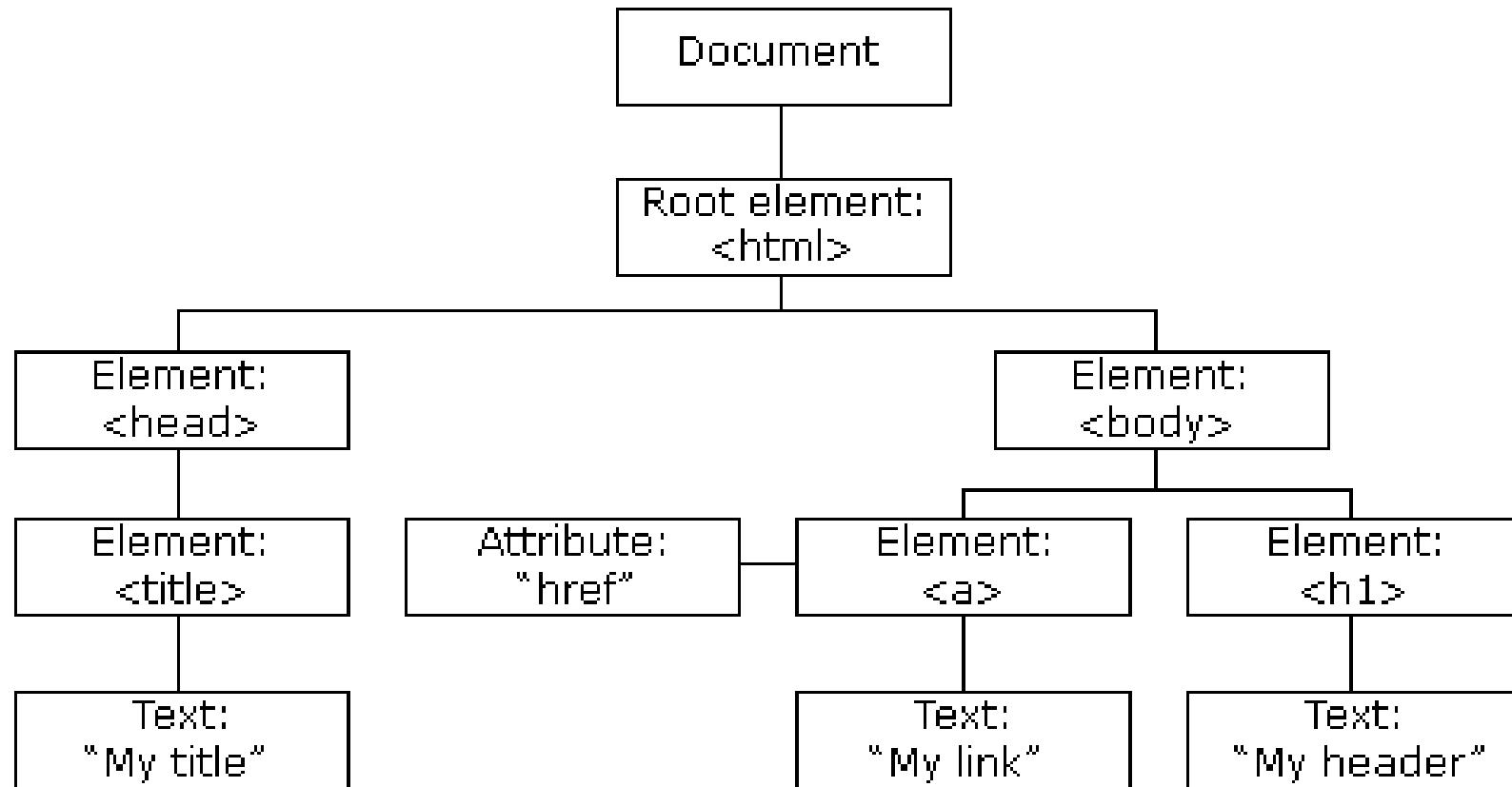
- Kao posljedicu ima sljedeće (neočekivano) ponašanje:

```
function f2() {
    c = 12; // globalna!!! varijabla jer nije deklarirana s var
}
```

JavaScript

- Za HTML dokument koji preglednik prikazuje u memoriji postoji izgrađeno stablo čvorova
- Svaki tag, tekst i komentar predstavljeni su zasebnim čvorom
- Čvor može imati atribute
- Govorimo o HTML DOM-u, odnosno **Document Object Model-u**

JavaScript: HTML DOM



JavaScript

- Vršni čvor JavaScriptu je dostupan kroz **window.document**. (**window.** se može izostaviti)
- Neka je x neki čvor. x nudi svojstva (naveden izbor):
 - x.parentNode → roditeljski čvor
 - x.childNodes → polje direktne djece
 - x.firstChild → prvo dijete
 - x.lastChild → zadnje dijete
 - x.attributes → polje atributa (a.name, a.value)

JavaScript

- Nad dokumentom možemo pozivati i metode koje pretražuju stablo čvorova:
 - X = **document.getElementByTagName("P")**; vraća polje svih čvorova koji predstavljaju tagove P u dokumentu (zgodno za pronaći sve linkove, sve slike, ...)
 - X = **document.getElementById("identifikator")**; pronalazi onaj tag koji je imao postavljen atribut id="identifikator" (taj bi morao biti jedinstven!)

JavaScript

- Imamo na raspolaganju timere
- Jednookidni:
window.setTimeout(funkcija, timeout);
po isteku timeout milisekundi jednom pozove zadalu funkciju
- Opetovani:
window.setInterval(funkcija, delta);
svakih delta milisekundi poziva funkciju

JavaScript

- Isječke JavaScript koda moguće je pisati i na razini tagova uz atribute koji opisuju različite događaje; npr.
`<form onsubmit=" .. tu ide javascript ..">`
`<input onclick=" ... ">`
- onclick, onchange, onmouseover, onmouseout, onkeydown, onkeyup, onsubmit, onload, ...
- http://www.w3schools.com/jsref/dom_obj_event.asp

JavaScript

- Spriječiti slanje formulara:
 - Izravno u tagu <form>:
`<form onsubmit="return false;">`
 - U gumbu submit:
`<input type="submit" onclick="...; return false;">`
- Validacija formulara:
 - Napišemo metodu npr. Validate() koja vraća true ako je sve OK, false inače, pa:
`<input type="submit" onclick="return validate()">`

Dinamičko modificiranje objekata

- U JavaScriptu, objektu se dinamički mogu dodavati svojstva i metode

```
let v = {};           // stvara novi objekt (preporuka je ne: new Object())
v.ime = "Pero";     // objektu dodaje svojstvo "ime"
v.prezime = "Perić"; // objektu dodaje svojstvo "prezime"
v.puno = function() {
    return this.ime + " " + this.prezime;
};

console.log( "Ime je " + v.ime + ", puno ime je " + v.puno() );
```

Vidi: http://www.w3schools.com/js/js_objects.asp

Dinamičko modificiranje objekata

- Objekte možemo stvarati i kraćom sintaksom koja podsjeća na definiranje mape (dajemo parove ključ: vrijednost)

```
let v = {  
    ime: "Pero",  
    prezime: "Perić",  
    puno: function() {  
        return this.ime + " " + this.prezime;  
    }  
};  
  
console.log( "Ime je " + v.ime + ", puno ime je " + v.puno() );
```

Legalno je i: `v["ime"]`

Dinamičko modificiranje objekata

- Možemo pisati i složenije strukture; npr. Polje od tri objekta kod kojih je svojstvo adresa novi objekt

```
let v = [
  {ime: "Pero", prezime: "Perić", adresa: {ulica: "Ilica 1", posta: 10000}},
  {ime: "Ana", prezime: "Anitić", adresa: {ulica: "Unska 3", posta: 10000}},
  {ime: "Jasna", prezime: "Jak", adresa: {ulica: "Slavonska 4", posta: 10000}}
]
console.log( v[2].adresa.ulica ); // Slavonska 4
```

AJAX



Ne taj...



Asynchronous Javascript And XML

AJAX

- AJAX = Asynchronous JavaScript and XML
- U klasičnom okruženju, korisnik klikanjem po linkovima i slanjem formulara traži nove HTML stranice
 - Da bi korisnik dobio nove podatke, mora zatražiti dohvati nove stranice
 - Često nepotrebno jer se nanovo generira (i učitava) zaglavlj stranice i niz zajedničkih sadržaja koji nemaju veze s podatcima

AJAX

- Ideja AJAX-a: dodati u JavaScript podršku za slanje GET/POST upita i prihvati rezultata
 - Na taj način JavaScript može **zatražiti samo podatke** u nekom prikladnom formatu (text, XML, JSON)
 - Odgovor dobiva JavaScript koji potom može **podatke prikazati manipulacijom DOM-a** u trenutnoj stranici, bez da ponovno učitava kompletну stranicu

AJAX

- Različiti preglednici pristup objektu preko kojeg je moguće obavljati AJAX pozive nude na različite načine

```
var xmlhttp;  
  
if (window.XMLHttpRequest) {  
    // code for IE7+, Firefox, Chrome, Opera, Safari  
    xmlhttp=new XMLHttpRequest();  
} else {  
    // code for IE6, IE5  
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
}
```

Vidi <http://www.w3schools.com/ajax/>

AJAX

- Na poslužitelju trebamo skripte koje dinamički generiraju tražene podatke (ali ih tipično **ne umataju u HTML** već u neki prikladniji format)
 - Čisti tekst
 - XML dokument
 - JSON
- Ovo ćemo pogledati na pripremljenoj web-aplikaciji koja je napisana u Javi
 - Tehnologija implementacije poslužitelja neovisna je o klijentskoj strani

AJAX

- AJAX pozivi mogu biti:
 - Sinkroni
 - dretva koja obavlja poziv će zablokirati dok ne stigne rezultat
 - Nezgodno jer može na jedno vrijeme smrznuti prikaz HTML stranice
 - Asinkroni
 - Prije poziva potrebno je definirati *callback* funkciju koja će biti pozivana za različite faze komunikacije

AJAX

- Objekt kojim se radi AJAX poziv ima članske varijable
 - *readyState*:
 - 0: još nije ništa započeto, pa sve do
 - 4: poziv je završen
 - *status*: sadrži HTTP status poziva (200 – OK)
 - *responseText*: podatci primljeni od poslužitelja kao tekst
 - *responseXML*: podatci primljeni od poslužitelja kao XML

AJAX

- Metodom `open()` definiramo parametre poziva (metoda: GET ili POST, adresa i URL parametri, sinkrono ili asinkrono)
- Metodom `send()` obavljamo sam poziv
- Ako je poziv asinkroni, prije toga trebamo registrirati `onreadystatechange` callback
- U sljedećem primjeru, servlet generira odgovor kao `text/plain`

AJAX

- Vraća se jedan redak koji predstavlja citat
- Tekst citata i naziv autora razdvojeni su znakom ljestvi (#)
- Možete izravno isprobati:

<http://localhost:8080/jsaplikacija/servlets/poruka>

AJAX

- Primjer:

```
xmlhttp.onreadystatechange = function() {
    if(xmlhttp.readyState==4 && xmlhttp.status==200) {
        var text = xmlhttp.responseText;
        var elems = text.split("#");
        document.getElementById("citat").innerHTML =
            elems[0] + "<br><b>" + elems[1] + "</b>";
    }
}

xmlhttp.open("GET", "servlets/poruka?dummy=" + Math.random(), true);

xmlhttp.send();
```

<http://localhost:8080/jsaplikacija/citati1.html>

JASON?





JSON

- Želimo li prenositi kompleksnije (tj. strukturirane) podatke, nekako ih trebamo formatirati
- U tom slučaju:
 - poslužitelj podatke koje je dohvatio iz baze mora zapisati u dogovorenom formatu (primjerice, u XML)
 - Klijent (naš JavaScript) mora primljeni tekst isparsirati u neku uporabivu strukturu (primjerice, stablo po kojem onda može trčati)

Vidi <http://json.org/>

JSON

- XML je portabilan format, ali je dosta nezgodan za uporabu
- Ideja: koristimo format koji je vrlo sličan načinu na koji JavaScript dozvoljava zapisivanje kompleksnih objekata (jedina razlika: i nazivi atributa pišu se pod navodnicima)
- U nastavku je primjer teksta po specifikaciji JSON; mime-tip: application/json

```
[  
  {"ime": "Pero", " prezime": "Perić", "adresa": {"ulica": "Ilica 1", "posta": 10000},  
   {"ime": "Ana", " prezime": "Anitić", "adresa": {"ulica": "Unska 3", "posta": 10000}},  
   {"ime": "Jasna", " prezime": "Jak", "adresa": {"ulica": "Slavonska 4", "posta": 10000}}]  
]
```

JSON vs XML

The image shows a screenshot of the Oxygen XML Editor interface, comparing two files: 'personal.xml' on the left and 'personal.json' on the right. Both files represent the same hierarchical data structure of personnel information.

personal.xml Content:

```
4 <personnel>
5   <person id="Big.Boss">
6     <name>
7       <family>Boss</family>
8       <given>Big</given>
9     </name>
10    <email>chief@oxygentools.com</email>
11    <link subordinates="one.worker"/>
12  </person>
13  <person id="one.worker">
14    <name>
15      <family>Worker</family>
16      <given>One</given>
17    </name>
18    <email>one@oxygentools.com</email>
19    <link manager="Big.Boss"/>
20  </person>
21  <person id="two.worker">
22    <name>
23      <family>Worker</family>
24      <given>Two</given>
25    </name>
26    <email>two@oxygentools.com</email>
27    <link manager="Big.Boss"/>
28  </person>
29  <person id="three.worker">
30    <name>
```

personal.json Content:

```
1  {"personnel": {"person": [
2    {
3      "id": "Big.Boss",
4      "name": {
5        "family": "Boss",
6        "given": "Big"
7      },
8      "email": "chief@oxygentools.com",
9      "link": {"subordinates": "one.worker"}
10    },
11    {
12      "id": "one.worker",
13      "name": {
14        "family": "Worker",
15        "given": "One"
16      },
17      "email": "one@oxygentools.com",
18      "link": {"manager": "Big.Boss"}
19    },
20    {
21      "id": "two.worker",
22      "name": {
23        "family": "Worker",
24        "given": "Two"
25      },
26      "email": "two@oxygentools.com",
27      "link": {"manager": "Big.Boss"}
28    }
29  ]}}
```

JSON - primjer

- Napravljen je servlet koji vraća polje slučajno odabralih citata; možete isprobati, pa pogledati na poslužitelju izvorni kod servleta:

<http://localhost:8080/jsaplikacija/servlets/poruke>

JSON na klijentu

- Jednom kada smo dobili takav tekst, JavaScript nudi funkciju `JSON.parse(text)` koja taj tekst konvertira izravno u strukturu JavaScript objekata

```
var text = primiJSONTekstOdPosluzitelja();
var polje = JSON.parse(text);
```

```
Console.log( polje[2].adresa.ulica ); // Slavonska 4
```

<http://localhost:8080/jsaplikacija/citati2a.html>

JSON na poslužitelju

- Na poslužitelju smo serijalizaciju Java objekata u JSON format morali raditi sami
- Napravili smo pri tome pogreške (kasnije ćemo ih pokazati)
- Postupak je mukotrpan jer bismo za složeniju strukturu morali rekurzivno raditi obilazak
- Bolje rješenje: koristiti neku od biblioteka koje će taj posao odraditi za nas

JSON na poslužitelju

- Postoji nekoliko popularnih biblioteka koje nude serijalizaciju/deserijalizaciju Java<->JSON
- Jackson:
<http://wiki.fasterxml.com/JacksonHome>
- Google-gson:
<http://code.google.com/p/google-gson/>
- Genson: <http://code.google.com/p/genson/>
- org.json: <http://www.json.org/java/>
- ...

JSON na poslužitelju

- Primjer uporabe gson biblioteke (Java!):

```
List<Quote> list =  
QuotesDB.getFilteredSelection(filter);  
Quote[] array = new Quote[list.size()];  
list.toArray(array);
```

```
resp.setContentType("application/json; charset=UTF-8");
```

```
Gson gson = new Gson();
```

```
String jsonText = gson.toJson(array);
```

```
resp.getWriter().write(jsonText);
```

```
resp.getWriter().flush();
```

Sam analizira strukturu,
otkriva nazive članskih
varijabli, ...

Složeniji primjer

- Radimo HTML stranicu koja sadrži kutiju za unos teksta preko koje korisnik može filtrirati citate prema prezimenu autora
- Filter se AJAX-om šalje na poslužitelj; tamo je servlet koji dohvaća polje citata i vraća ga u JSON formatu klijentu
- Timeout je postavljen na 5 sekundi (da se vidi što se događa)
- Zatražite A pa Al pa D

Složeniji primjer

- Po isteku vremenskog ograničenja pozvat ćemo metodu koja će AJAX-om zatražiti od servleta popis filtriranih citata

`http://localhost:8080/jsaplikacija/servlets/porukef?filter=Alb`

VratiPorukeFServlet

`http://localhost:8080/jsaplikacija/citati3b.html`

Složeniji primjer

- Zatražite kao filter: M
- Možete li objasniti što se dogodilo?

Složeniji primjer

- Imamo sigurnosni problem poznat pod nazivom *script-injection!*
- Da bismo ga riješili, nužno je pravilo *escapeati* sve što umećemo u HTML

jQuery

- Za postizanje bilo kakvih složenijih funkcionalnosti, često je potrebno pisati puno linija koda u JavaScriptu
- Za mnoštvo funkcionalnosti, moramo pisati kôd koji ispituje u kojem se pregledniku izvodi pa prilagođava JavaScript koji poziva (sjetite se načina dobivanja AJAX objekta)

jQuery

- jQuery je JavaScript-biblioteka koja:
 - Definira skup univerzalnih funkcionalnosti koje nudi korisnicima (sama se brine da to radi u svim preglednicima)
 - Nudi API više razine u odnosu na dostupan u osnovnom JavaScriptu
 - Pišemo manje linija koda za istu funkcionalnost
 - Nudi još niz dodanih naprednih funkcionalnosti (rad s dijalozima, različite kontrole, animacije, ...)

jQuery

- Mi ćemo pogledati samo osnovnu funkcionalnost:
 - Pretraživanje i jednostavna manipulacija DOM-a
 - Uporaba AJAX-a
- Uključivanje
 - skinuti lokalno potrebnu .js datoteku (bolje ne)
 - Koristiti CDN (Content Delivery Network): linkati biblioteku direktno s Interneta (pospješuje dijeljenje, smanjuje vrijeme potrebno za učitavanje stranice)

CSS



HTML



HTML
+
CSS

CSS

- *Cascading Style Sheets* (verzija 3?)
- U neka davna vremena, HTML specifikacija bavila se i sadržajem i načinom njegova prikaza
 - Definiranje logičke strukture:
 - Tagovi H1-H6, P, A, IMG, TABLE, OL/DL/UL, ...
 - Definiranje načina prikaza:
 - Tagovi FONT, CENTER, B, I, U, PRE, ...
 - I jedno i drugo:
 - Tagovi CODE, BLOCKQUOTE

Vidi <http://www.w3schools.com/css/default.asp>

CSS

- Moderne web-aplikacije prikaz žele moći dinamički mijenjati
 - To znači da prikaz ne smije biti određen strukturom tagova
- Ideja je u HTML dokumentu koristiti tagove koji definiraju logičku strukturu te zasebne odjeljke općenitim tagovima DIV i SPAN
- Semantiku informacije koju predstavlja TAG opisati atributom **class**

CSS

- Potom nezavisno definirati način prikaza i stilizacije sadržaja, na način da se definiraju filtri koji selektiraju tagove te atributi prikaza koji se primjenjuju na selektirane tagove
- Kao filter se može koristiti
 - ime taga (P će selektirati sve tagove P u dokumentu)
 - naziv razreda (sintaksa je točka pa razred; npr. .adresa)
 - Identifikator taga (ljestve pa identifikator, #citat)

CSS

- Moguće je koristiti i
 - lanac: npr. “TABLE P” će selektirati sve P tagove koji su negdje u tablicama (imaju kao pretka u stablu tag TABLE)
 - Tag s razredom: “TABLE DIV.adresa” će selektirati samo one DIV tagove koji imaju atribut class postavljen na “adresa” i koji imaju kao pretka tag TABLE
 - Stanja: “INPUT:FOCUS” selektira sve INPUT tagove koji su fokusirani (valjda samo jedan?)

CSS

- Nakon filtra u vitičastim se zagradama definira stil, po formatu “ključ: vrijednost;”
- Evo primjera:

```
h1 {font-style: italic;}  
h1:HOVER {font-style: italic; color: red;}
```

- U stranici stil možemo (preporuka: u zaglavlju):
 - Definirati unutar <style>...</style>
 - Uključiti izvana:
`<link rel="stylesheet" type="text/css" href="stil1.css">`

CSS

- Stil možemo definirati i lokalno u tagu kroz atribut **style**
 - Tada se stil odnosi samo na taj konkretan tag
 - Takva uporaba se ne preporuča
 - Primjer:
`<p style="color: red; font-style: italic;"> ... </p>`

CSS

- Svaki citat u stranicu uključujemo ovako:

```
<div class='quoteBox'>
  <div class='quoteText'> ... tu ide tekst ... </div>
  <div class='quoteAuthor'> ... tu ide autor ... </div>
</div>
```

- Stoga stilove pišemo ovako:

```
.quoteBox { margin-top: 10px; ... display: inline-block; }
.quoteText { padding-left: 5px; ... overflow: auto; }
.quoteAuthor { padding-right: 5px; ... font-variant: small-caps; }
```



REST

REST

- REST je kratica od:
Representational State Transfer
 - Način oblikovanja web-aplikacija
- Utemeljen na nekoliko načela
 - 1. sve što se mora moći identificirati na webu treba tretirati kao resurs
 - 2. koriste se standardne metode HTTP-a za rad s resursima

REST



POST vs PUT:

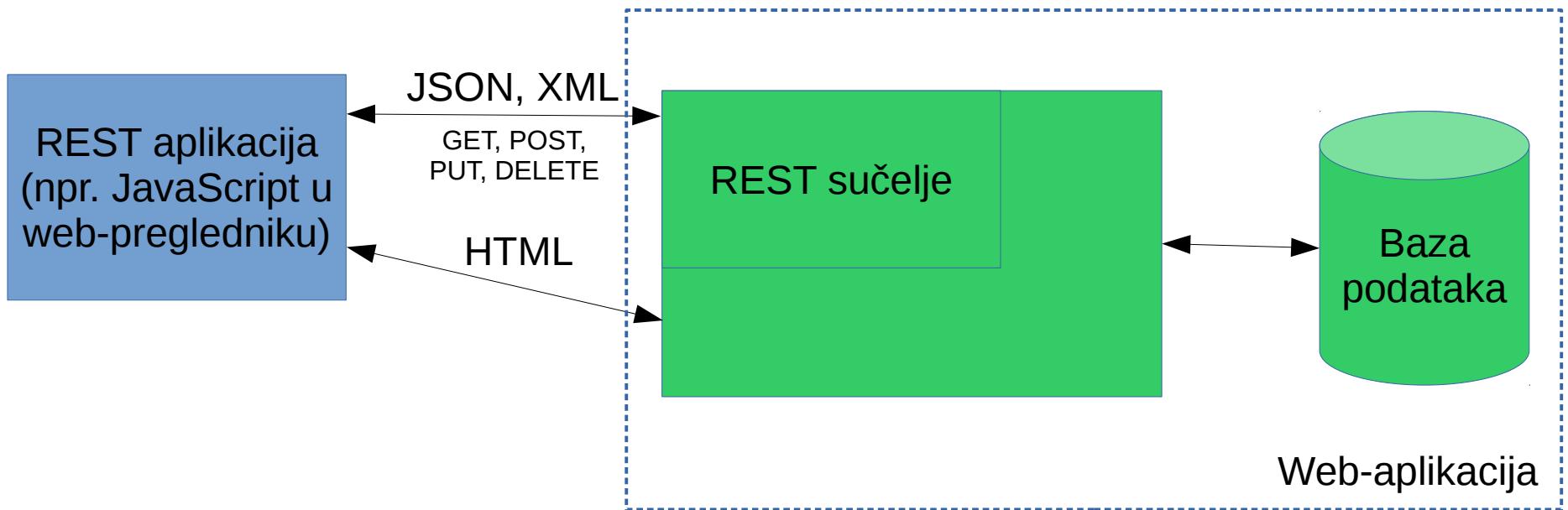
<https://stackoverflow.com/questions/630453/put-vs-post-in-rest>

REST

Resource	POST create	GET read	PUT update	DELETE delete
/user	Stvara novog korisnika	Dohvaća sve korisnike	Masovno ažuriranje korisnika	Brisanje svih korisnika
/user/1234	Pogreška (stvaranje korisnika koji već postoji)	Dohvaća korisnika čiji je zadan identifikator	Ažuriranje podataka za zadanog korisnika (ako postoji), inače pogreška	Brisanje zadanog korisnika (prema identifikatoru)

- REST dozvoljava da se resurs vrati u različitim oblicima
 - Primjerice, kao XML, JSON, ...
- Status operacije javlja kroz HTTP (200, 404, ...)

REST



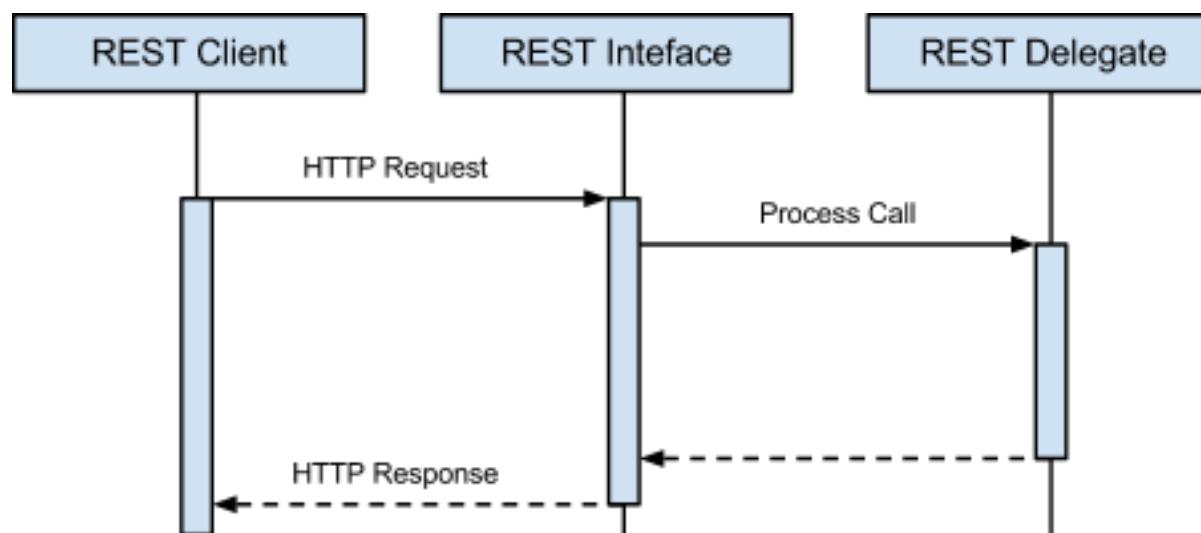
- Ideja: omogućiti klijentskoj aplikaciji da učita osnovnu stranicu i potom podatke dohvaća i ažurira REST-om (AJAX, JSON/XML)

REST

- Podršku na poslužitelju za REST možemo pisati samostalno na razini servleta
 - Nešto slično radili ste u aplikaciji za blog; sjetite se kakvi su bili URL-ovi
- Život si možemo olakšati ako uzmemos neku od gotovih biblioteka

REST

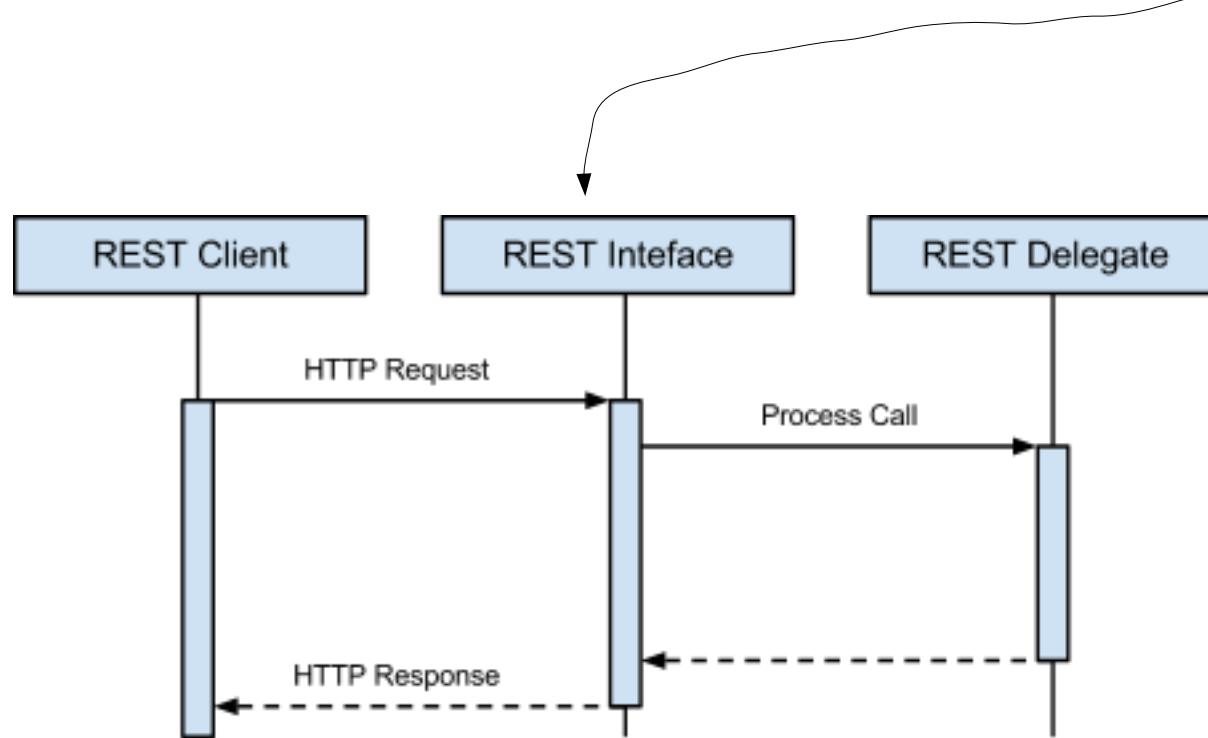
- U demonstracijskoj aplikaciji iskorištena je biblioteka jersey koja predstavlja implementaciju JSR-339 specifikacije (JAX-RS 2.0)
- Model:



REST

Servlet `org.glassfish.jersey.servlet.ServletContainer`

Potrebno ga je mapirati na neku početnu stazu unutar koje nastaje prostor resursa; to radimo u `web.xml`; kod nas će to biti `/rest/` (što je lokalno unutar konteksta web-aplikacije)

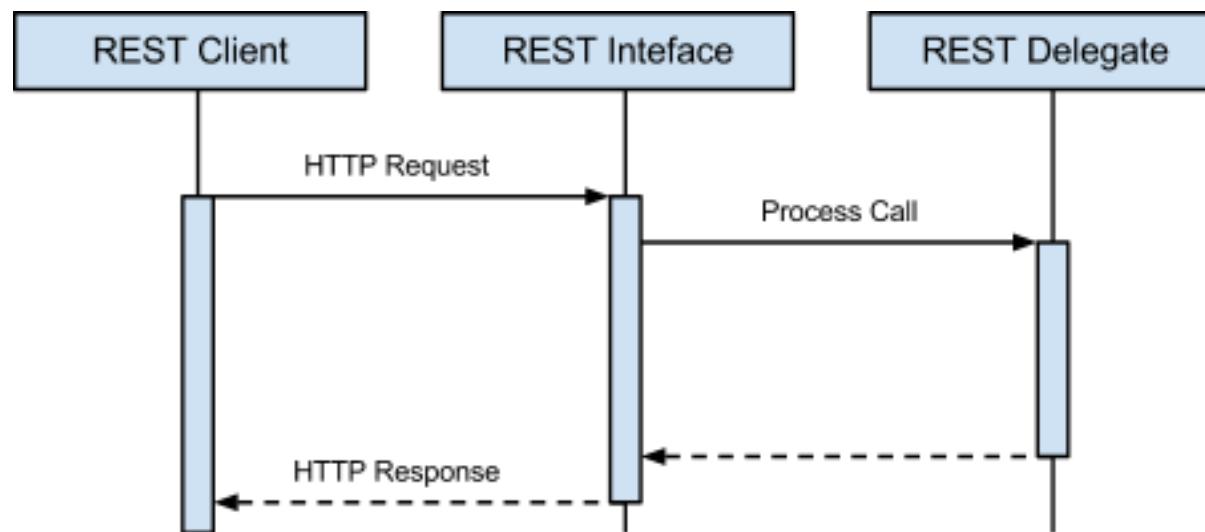


REST

Potom definiramo resurse. Kod nas ćemo i dalje raditi s citatima.

Razredi su hr.fer.zemris.jsdemo.rest.QuoteXML i QuoteJSON

Jerseyu za svaki resurs treba reći na kojem relativnom URL-u je dostupan (imat ćemo /quotex i /quotej), i potom mapirati metode na parove HTTP metoda zahtjeva + podURL



REST

`http://localhost:8080/jsaplikacija/rest/quotex`

Staza do web-aplikacije

Staza do jerseyevog servleta koji radi *dispatching*

Staza kojom tražimo ispis broja citata u formatu XML

REST

`http://localhost:8080/jsaplikacija/rest/quotex/4`

Staza do web-aplikacije

Staza do jerseyevog servleta koji radi *dispatching*

Staza kojom tražimo dohvati četvrtog citata u XML obliku

REST

`http://localhost:8080/jsaplikacija/rest/quotej`

Staza do web-aplikacije

Staza do jerseyevog servleta koji radi *dispatching*

Staza kojom tražimo ispis broja citata u formatu JSON

REST

`http://localhost:8080/jsaplikacija/rest/quotej/4`

Staza do web-aplikacije

Staza do jerseyevog servleta koji radi *dispatching*

Staza kojom tražimo dohvati četvrtog citata u JSON obliku

REST

- Pogledati kod spomenutih razreda
 - Uočiti kako se anotacijama definira što je koji resurs i koja se metoda poziva kada
 - Prikazani primjeri namjerno definiraju dva resursa (koja su oba isti citati, što nije dobro, ali je lakše pratiti kod)
 - U stvarnoj primjeni, metoda deklarira koje sve oblike može generirati i vraća sam objekt koji se potom automatski konvertira u taj oblik

REST

- Pogledati kod spomenutih razreda
 - Uočiti kako se anotacijama definira koji se dio URL-a tretira kao parametar te kako se taj parametar metodi predaje kroz argument (i koji)
 - Više uopće ne radimo s razredima HttpServletRequest i HttpServletResponse niti prolazimo kroz parametre i radimo konverzije tipova

REST

- Za konverzije za koje ne postoje ugrađeni konvertori, dovoljno je napisati razred koji zna obaviti konverziju
 - Razred je potrebno anotirati s `@Provider` i tipom koji producira, treba implementirati sučelje `MessageBodyWriter` i ponuditi implementacije potrebnih metoda
 - Prilikom bootanja, Jersey će pronaći sve takve anotirane konverte i automatski ih dalje koristiti

GOTOVI SMO!

Još neki linkovi

- // Uvod u DOM:

<http://www.quirksmode.org/dom/intro.html>

http://www.w3schools.com/js/js_htmldom.asp

- <http://stackoverflow.com/questions/6585971/best-practice-for-using-dom-level-0-in-modern-javascript-applications>

- <http://www.quirksmode.org/js/dom0.html>

- https://en.wikipedia.org/wiki/DOM_events

- // JavaScript objektni model:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

Još neki linkovi

- JSON:
 - * Uvod:
<http://www.json.org/>
 - * Ideja:
<http://www.oracle.com/technetwork/articles/java/json-1973242.html>
- GSON:
<https://code.google.com/p/google-gson/>
- Usporedba:
<http://www.developer.com/lang/javascript/top-7-open-source-json-binding-providers-available-today.html>