



Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM II

Predavanje 11 - JSON, Ajax

4. lipnja 2018.

Sastavio: Zvonimir Bujanović



JSON - JavaScript Object Notation

- Služi za reprezentaciju JavaScript objekata u obliku stringa (niza byteova).
 - String se može spremiti na disk.
 - String se može poslati kroz mrežu.
- JSON omogućava i konverziju u suprotnom smjeru: string se može vrlo lako "raspakirati" u originalni JavaScript objekt.
- Proces konverzije objekta u format (niz byteova) pogodan za spremanje na disk ili slanje kroz mrežu se zove **serijalizacija** ili *marshalling*.
- Proces konverzije niza byteova natrag u objekt se zove **deserijalizacija** ili *unmarshalling*.

Pretpostavimo da smo deklarirali varijablu

```
1 var studenti = [ {  
2     JMBAG: "1234567890",  
3     ime: "Pero Perić",  
4     ocjene: [5, 3, 2]  
5 }, {  
6     JMBAG: "0987654321",  
7     ime: "Ana Anić",  
8     ocjene: [2, 4]  
9 } ];
```

- Dobivanje reprezentacije objekta u obliku stringa:

```
1 var jsonStudenti = JSON.stringify( studenti );
```

- Dobiveni string je:

```
'["JMBAG":"1234567890","ime":"Pero  
Perić","ocjene":[5,3,2],"JMBAG":"0987654321","ime":"Ana  
Anić","ocjene":[2,4]]'
```

Pretpostavimo da imamo string

```
jsonStudenti = ' ["JMBAG":"1234567890","ime":"Pero  
Perić","ocjene":[5,3,2],"JMBAG":"0987654321","ime":"Ana  
Anić","ocjene":[2,4]] '
```

- Iz tog stringa možemo dobiti JavaScript objekt:

```
1 var s = JSON.parse( jsonStudenti );
```

- Varijabla `s` se sad može koristiti na uobičajen način:

```
1 $( "#p2" ).append(  
2     "<br />JMBAG: " + s[0].JMBAG + " ime: " + s[0].ime +  
3     "<br />JMBAG: " + s[1].JMBAG + " ime: " + s[1].ime );
```

- Funkcije za generiranje i parsiranje JSON stringova postoje i u drugim prog. jezicima.
- Ograničenja:
 - Ne mogu se kodirati funkcije, regularni izrazi.
 - String-reprezentacija objekata se ne konvertira automatski u odgovarajući tip sa svim članskim funkcijama (npr. `Date`)
- Drugo ograničenje možemo popraviti ovako: ako je originalni JavaScript objekt imao člana `datumRodjenja` tipa `Date`, onda:

```
1 var s = JSON.parse( jsonStudenti, function(key, value)
2 {
3     if( key === "datumRodjenja" )
4         return new Date( value );
5     else
6         return value;
7 } );
```

Primjer 1 pokazuje:

- Konverziju iz JavaScript objekta u JSON string.
- Konverziju natrag iz JSON stringa u JavaScript objekt.
- Za neke objekte (tipično, ako pripadna klasa ima funkcije članice), trebamo prilagoditi konverziju iz JSON stringa prilikom poziva `JSON.parse`. Najčešće treba samo pozvati odgovarajući konstruktor.

- U PHP-u:
 - `JSON.stringify()` \rightsquigarrow `json_encode()`;
 - `JSON.parse()` \rightsquigarrow `json_decode()`.
- U nastavku će nam trebati PHP skripte koje ne generiraju HTML nego JSON. To se postiže pomoću specijalnog *headera*:

```
1 function sendJSONandExit( $message )
2 {
3     // Kao izlaz skripte pošalji $message u JSON formatu i
4     // prekini izvođenje.
5     header( 'Content-type:application/json;charset=utf-8' );
6     echo json_encode( $message );
7     flush();
8     exit( 0 );
9 }
10
11 $message = [];
12 $message[ 'JMBAG' ] = $JMBAG; $message[ 'ocjene' ] = $ocjene;
13 sendJSONandExit( $message );
```

Ajax - Asynchronous JavaScript + XML

- Tehnika za dohvaćanje dodatnih podataka sa servera bez ponovnog učitavanja web-stranice.
- Dohvaćanje je **asinkrono**, tj. za vrijeme komunikacije sa serverom je web-stranica i dalje respozivna.
- Omogućava dohvaćanje podataka bez obzira na format (dakle, ne nužno u XML formatu).
- Danas se najčešće koristi u kombinaciji sa JSON-om.
- Radi potpunosti, opisat ćemo prvo kako se koristi bez biblioteke jQuery. Poslije ćemo Ajax koristiti isključivo pomoću jQuery jer je bitno jednostavnije.
- Novija tehnologija je **Fetch API**, koji daje više mogućnosti i to na jednostavniji način.

- Stvaranje novog zahtjeva prema serveru:

```
1 var xhr = new XMLHttpRequest();  
2 xhr.open( tipZahtjeva, adresaNaServeru, true );
```

Tip zahtjeva je "GET" ili "POST".

- GET – tipično se koristi za upit serveru, u slučaju kad server ne pohranjuje podatke koje mu šaljemo. Pogodan za kraće upite.

```
1 xhr.open( "GET", "getinfo.php?user=Ana&age=21", true );  
2 xhr.send();
```

- POST – tipično se koristi za slanje podataka koje server treba pohraniti. Pogodan i za dulje poruke. Malo složenije, ali moćnije no sa GET:

```
1 xhr.open( "POST", "getinfo.php", true );  
2 xhr.setRequestHeader( "Content-type",  
3                       "application/x-www-form-urlencoded" );  
4 xhr.send( "user=Ana&age=21" );
```

Poziv je asinkron. Zato se definira *callback* funkcija kojom reagiramo na događaj `readystatechange`, tj. promjenu stanja XHR objekta.

```
1 xhr.onreadystatechange = function() {  
2     if( xhr.readyState == 4 && xhr.status == 200 ) {  
3         // xhr.responseText = string koji sadrži serverov odgovor  
4     }  
5 }
```

Korisna stanja `xhr.readyState`:

- 3: answer in process – korisno za indikator napretka
- 4: finished – obično samo koristimo samo ovo stanje

`xhr.status` je rezultat HTTP zahtjeva:

- 200 = OK
- 304 = dohvaćeno iz browserovog cache-a
- 404 = adresa nije pronađena, itd.

Korištenje Ajax-a pomoću biblioteke jQuery je jednostavnije:

- GET:

```
1 $.get(  
2     "getinfo.php", // Skripta koja obrađuje podatke.  
3     {  
4         // Podaci koji se šalju serveru. Dobit će ih u $_GET.  
5         user: "Ana",  
6         age: 21  
7     },  
8     function( data, status )  
9     {  
10        if( status === "success" )  
11        {  
12            // Ovdje ide kod u slučaju da je server uspješno  
13            // vratio odgovor. Odgovor se nalazi u varijabli data.  
14        }  
15    }  
16 );
```

- Puno kontrolu daje funkcija `.ajax`:

```
1 $.ajax(  
2 {  
3     url: "getinfo.php",  
4     data:  
5     {  
6         user: "Ana",  
7         age: 21  
8     },  
9     type: "GET",  
10    dataType: "json", // očekivani povratni tip podatka  
11    success: function( json ) { ... },  
12    error: function( xhr, status, errorThrown ) { ... },  
13    complete: function( xhr, status ) { ... }  
14 } );
```

- Ima još puno naprednijih opcija.
- `dataType` = "json", "html", itd. (default: *intelligent guess*)
- `type` = "GET", "POST" (default: GET)

Primjer 2 - "Suggest"

Server sugerira mogući izbor imena na temelju onog što je korisnik utipkao i svoje liste poznatih imena.

Unesi svoje ime:

- Maja
- Marko
- Mirko

Klijentski dio – suggest.html

```
1 txt.on( "input", function(e) {  
2     var unos = $( this ).val();  
3  
4     $.ajax( {  
5         url: "suggest.php",  
6         data: { q: unos },  
7         success: function( data ) {  
8             $( "#datalist_imena" ).html( data );  
9         } );  
10    } );
```

Primjer 2 - "Suggest"

Serverski dio – suggest.php

```
1 <?php
2 $imena = [ "Ana", "Ante", "Boris", "Maja", "Marko",
3           "Mirko", "Slavko", "Slavica" ];
4 $q = $_GET[ "q" ];
5
6 foreach( $imena as $ime )
7     if( strpos( $ime, $q ) !== false )
8         echo "<option value='" . $ime . "' />\n";
9 ?>
```

Ajax - Debugiranje

- Debugiranje Ajax-a je prilično nezgrapno.
- Ako je greška u skripti na serveru, on neće niti uspjeti poslati poruku, pa nećemo znati što je krivo (npr. imamo syntax error u PHP-u).
- Opcije:
 - Vrlo defanzivno programiranje serverske strane.
 - Server treba slati detaljne poruke o greškama.
 - Korištenje Web Developer alata u Firefoxu (tab Network, Response).
 - Ili: direktan pristup adresi PHP skripte, npr. `suggest.php?q=M`.

Network - Primjer 3 - Chat

Status	Method	File	Domain	Type	Headers	Cookies	Params	Response
200	GET	Primjer 3 - Chat.html	rp2.studenti.math.hr	html	Filter properties			
200	GET	jquery.js	ajax.googleapis.com	js	JSON			
200	GET	Primjer 3 - Chat.php?timestamp=0&cache=1464543615478	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543510&cache=14645...	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?ime=Mirko&msg=Hello.	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543622&cache=14645...	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?ime=Mirko&msg=Kako%20si?	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543627&cache=14645...	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543675&cache=14645...	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?ime=Mirko&msg=A%20ti?	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543680&cache=14645...	rp2.studenti.math.hr	html				
200	GET	Primjer 3 - Chat.php?timestamp=1464543685&cache=14645...	rp2.studenti.math.hr	html				

Response: `msg: "Slavko: Hm."`
`timestamp: 1464543685`

Zadatak 1

Napravite jednostavni kalkulator (4 osnovne računske operacije).

- Rezultat se izračunava na serveru.
- Da bi se rezultat prikazao ne treba ponovno učitati cijelu stranicu.
- Dakle, klikom na računsku operaciju, rezultat se dohvaća Ajax pozivom.

10	20	+	-	*	/	0.5
----	----	---	---	---	---	-----

Na web-stranici nalazi se nekoliko linkova klase `fileLink` na datoteke koje se nalaze na serveru. U zaglavlju (`head`) stranice je include-ana skripta `zadatak2.js`.

Bez izmjene HTML-a, napravite sljedeće:

- Kada korisnik prijeđe mišem iznad linka klase `fileLink`, iznad linka se treba pojaviti "balon" u kojem piše veličina te datoteke i vrijeme zadnje modifikacije.

Uputa:

- Reagirajte na događaje `mouseenter` i `mouseleave`.
- Na `mouseenter` se Ajaxom dohvaćaju odgovarajuće informacije o datoteci sa servera.

Opisali smo ovaj slučaj:

- 1 Klijent inicira kontakt prema serveru.
- 2 Klijent šalje upit/podatke serveru.
- 3 Server na temelju upita/podataka ima odmah spreman odgovor i odmah ga šalje klijentu.

Kako riješiti sljedeći slučaj?

- 1 Na serveru će se u nekom trenutku pojaviti podaci koje želi poslati klijentu.
- 2 Klijent treba stalno biti spreman primiti te podatke.

Tipičan scenario: chat, igra na poteze za dva igrača, gmail.

Jedna obično loša ideja (*short polling*): klijent svake sekunde provjerava jesu li podaci spremni.

```
1 setInterval( napraviUpit, 1000 );
```

Long polling (Comet)

- ❶ Klijent otvori XHR zahtjev prema serveru.
- ❷ Server ne odgovara na zahtjev sve dok nema spremne podatke:
 - XHR veza se ne prekida sve dok server ne generira cijelu stranicu.
 - Server ima beskonačnu petlju u kojoj provjerava jesu li podaci spremni.
 - Često se unutar petlje stavi da server "spava" kako se ne bi radilo stalno opterećenje njegovog procesora.
- ❸ Kada su podaci spremni, server ih pošalje klijentu i prekine vezu.
- ❹ Klijent primi podatke i ponovno uspostavlja vezu sa serverom.

Mana ovog pristupa:

- Server ima uspostavljenu vezu prema klijentu sve dok mu ne pošalje podatke.

Web-aplikacija omogućuje chat za sve korisnike koji se spoje na stranicu.

Klijentski dio (JavaScript):

- Pomoću *long polling-a* se očekuju podaci od skripte `cekajPoruku.php`.
 - Ajax upit GET-om pošalje `timestamp` (vrijeme kad je zadnji put primljena poruka), `cache` (trenutno vrijeme).
 - Podaci od servera će stići u JSON formatu.
 - JSON objekt će imati definirana polja `msg` (sadržaj poruke) i `timestamp` (vrijeme kad je poruka poslana sa servera).
 - Kad dobije poruku od servera, doda `msg` na kraj div-a i ide ponovno na prvu točku.
- Kada korisnik klikne na "Pošalji":
 - Pomoću Ajax-a se kontaktira skripta `posaljiPoruku.php`.
 - GET-om se pošalju stringovi `ime` (ime korisnika koji šalje poruku) i `msg` (sadržaj poruke).

Web-aplikacija omogućuje chat za sve korisnike koji se spoje na stranicu.

Serverski dio (PHP):

- `posaljiPoruku.php`:
 - Iz GET-a pročita stringove `ime` (ime korisnika koji šalje poruku) i `msg` (sadržaj poruke).
 - Zapiše te stringove u datoteku `chat.log`.
- `cekajPoruku.php`:
 - Iz GET-a pročita `timestamp` (vrijeme kad je zadnji put primljena poruka), `cache` (trenutno vrijeme).
 - Svaki 10ms pogleda vrijeme zadnje modifikacije datoteke `chat.log`.
 - Ako je to vrijeme veće od `timestamp`, generira JSON objekt s poljima `msg` (sadržaj datoteke) i `timestamp` (vrijeme zadnje modifikacije datoteke).
 - Ispiše taj JSON objekt.

Zadatak 3

- U svojoj bazi na rp2-serveru napravite tablicu **Dionice** sa stupcima **Oznaka**, **Ime**, **Cijena**, te dodajte nekoliko redaka u nju.
- Napravite web-stranicu **zadatak3.html** koja dinamički dohvaća i prikazuje podatke iz te tablice:
 - Koristeći long-polling, preko JavaScripta kontaktirajte skriptu **zadatak3.php**.
 - Ta skripta neka vraća podatke u JSON formatu (polje).
 - Vraćene podatke prikažite u HTML tablici generiranoj JavaScript-om.
- Preko **phpmyadmin** dodajte novu dionicu u bazu ili promijenite cijenu nekoj postojećoj. Bez manualnog osvježavanja stranice **zadatak3.html**, uvjerite se će podaci biti automatski osvježeni čim ih izmijenite u MySQL bazi.
- **Uputa.** (U novijim verzijama MySQL-a radi i prihvaćeni odgovor.)

- Ranije je adresa skripte koju kontaktiramo preko Ajax-a morala biti na istoj domeni kao i HTML.
- Danas je to ograničenje prevaziđeno (vidi **CORS**).
- Popularne su i druge tehnike za istu svrhu, poput **JSONP**.
 - Na primjer, moguće je dohvatiti **tweetove** koristeći ovu tehniku. Vidi **detalje**.
- Alternativa long-polling je novija tehnika **Server-sent events (SSE)**.