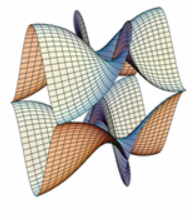


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM II

Predavanje 11 - JSON, Ajax

31. svibnja 2023.

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*.

JSON

Pretpostavimo da smo deklarirali varijablu

```
let studenti = [ {
    JMBAG: '1234567890',
    ime: 'Pero Perić',
    ocjene: [5, 3, 2]
}, {
    JMBAG: '0987654321',
    ime: 'Ana Anić',
    ocjene: [2, 4]
} ];
```

- Dobivanje reprezentacije objekta u obliku stringa:

```
let json_studenti = JSON.stringify( studenti );
```

- Dobiveni string je:

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

JSON

Pretpostavimo da imamo string

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

- Iz tog stringa možemo dobiti JavaScript objekt:

```
let s = JSON.parse( json_studenti );
```

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

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

JSON

- 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 `datum_rodjenja` tipa `Date`, onda:

```
let s = JSON.parse( json_studenti, function(key, value)
{
    if( key === 'datum_rodjenja' )
        return new Date( value );
    else
        return value;
} );
```

Primjer 1

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.

JSON i PHP

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

```php
function sendJSONandExit( $message )
{
    // Kao izlaz skripte pošalji $message u JSON formatu i
    // prekini izvođenje.
    header( 'Content-type:application/json;charset=utf-8' );
    echo json_encode( $message );
    flush();
    exit( 0 );
}

$message = [];
$message[ 'JMBAG' ] = $JMBAG; $message[ 'ocjene' ] = $ocjene;
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 responzivna.
- Omogućava dohvaćanje podataka bez obzira na format (dakle, ne nužno u XML formatu).
- Danas se najčešće koristi u kombinaciji sa JSON-om.

- Bez biblioteke jQuery, Ajax možemo koristiti pomoću:
  - XMLHttpRequest – starija i nespretna verzija.
  - Fetch API – moderna verzija koja koristi tzv. Promise.
- jQuery također omogućava jednostavno korištenje Ajax-a.

Promise

- Ako se nešto izvršava asinkrono (npr. dohvat podataka sa servera), *promise* omogućava da se neki kod izvrši po završetku.
- Ispis donjeg koda je: `"prije poslije"` i onda za 5 sekundi `"gotovo"`.

```
let promise = new Promise( function(resolve, reject) {
    // Ova (anonimna) funkcija se odmah počne izvršavati.
    // setTimeout je asinkron i izlazimo iz anonimne funkcije.
    // Za 5 sekundi se "promise" završi s rezultatom "gotovo".
    setTimeout( function() { resolve( 'gotovo' ); }, 5000 );
});

console.log( 'prije' );
promise.then( function(result) {
    // Kod koji se izvrši kada se "promise" uspješno završi.
    console.log( 'Uspješno riješen promise: ' + result );
} );
console.log( 'poslije' );
```

Promise - async/await

async/await

- Ekvivalentnu, a jednostavniju sintaksu omogućava par **async/await**.
- "Čekanje" promise-a umjesto sa **then** radimo sa **await**.
- Funkcija u kojoj čekamo (await) nešto asinkrono mora biti označena sa **async**.
- Takve funkcije automatski vraćaju promise.

```
1  async function f() {
2      let promise = new Promise( function(resolve, reject) {
3          setTimeout( function() { resolve( 'gotovo' ); }, 5000 );
4      });
5
6      let result = await promise; // Stoji dok se promise ne završi.
7      console.log( result );
8  }
9
10 console.log( 'prije' ); f(); console.log( 'poslije' );
```

Fetch API

- Funkcija `fetch` asinkrono dohvaća podatke sa zadanog URL-a.
- Vraća *promise* čijim uspješnim završetkom dobivamo podatke.
- Donji primjer šalje PHP skripti podatke pomoću GET.

```
1  async function get_data()
2  {
3      // Ajax poziv sa GET: parametri se šalju kroz URL.
4      let response = await fetch( 'getinfo.php?user=Ana&age=21' );
5
6      // Dohvatimo podatke iz response-a.
7      let data = await response.text();
8
9      // Ako su podaci u JSON formatu, onda:
10     // let data = await response.json();
11     // Sad tu nešto radimo s podacima iz data...
12 }
13
14 get_data();
```

Fetch API

- Ako serveru šaljemo podatke pomoću POST ⇝ opcije.

```
async function post_data()
{
    // Ajax poziv sa POST: parametri se šalju kroz opciju body.
    let response = await fetch( 'getinfo.php', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json;charset=utf-8'
        },
        body: JSON.stringify( { user: 'Ana', age: 21 } )
    } );

    // Dohvatimo podatke iz response-a.
    let data = await response.text(); // ili .json()
}

post_data();
```

Fetch API

- Neuspjeli *await* automatski baca iznimku.
- Greške možemo detektirati okruživanjem `await` sa `try/catch` i/ili hvatanjem iznimke pomoću `catch` u pozivu `async` funkcije.

```
async function loadJson(url) {
    let response = await fetch(url);

    if (response.status == 200) {
        // HTTP status 200 = uspješno dohvaćeni podaci.
        let json = await response.json();
        console.log( json );
        return;
    }

    // HTTP status nije 200 -> greška.
    throw new Error(response.status);
}

loadJson( 'getinfo.php' ).catch(alert);
```

jQuery - Ajax

Korištenje Ajax-a pomoću biblioteke jQuery je jednostavnije:

- GET:

```
$.get(
  'getinfo.php', // Skripta koja obrađuje podatke.
  {
    // Podaci koji se šalju serveru. Dobit će ih u $_GET.
    user: 'Ana',
    age: 21
  },
  function( data, status )
  {
    if( status === 'success' )
    {
      // Ovdje ide kod u slučaju da je server uspješno
      // vratio odgovor. Odgovor se nalazi u varijabli data.
    }
  }
);
```

jQuery - Ajax

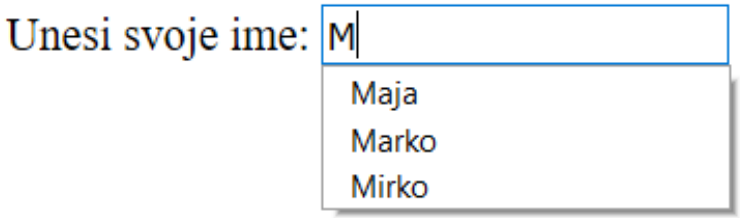
- Punu kontrolu daje funkcija .ajax:

```
1  $.ajax(
2  {
3      url: 'getinfo.php',
4      data:
5      {
6          user: 'Ana',
7          age: 21
8      },
9      method: '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*)
- `method` = `'GET'`, `'POST'` (default: `'GET'`)

Primjer 2 - "Suggest"

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



Klijentski dio – `suggest.html`

```
txt.on( 'input', function(e) {
  let unos = $( this ).val();

  $.ajax( {
    url: 'suggest.php',
    data: { q: unos },
    success: function( data ) {
        $( '#datalist_imena' ).html( data );
    } );
} );
```

Primjer 2 - "Suggest"

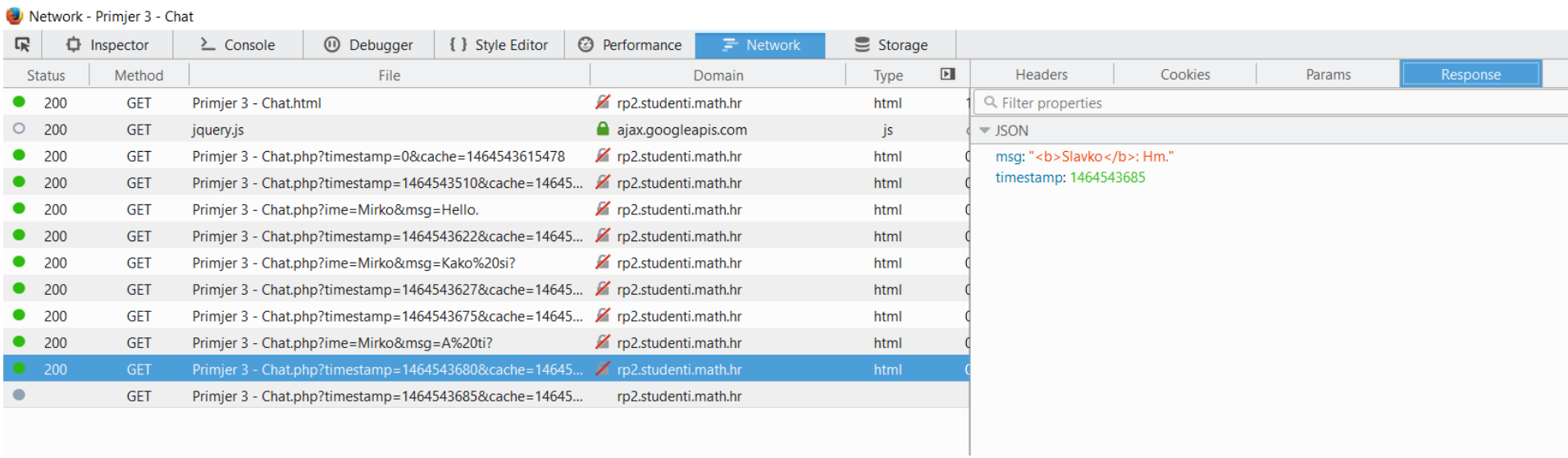
Serverski dio – `suggest.php`

```php
<?php
$imena = [ 'Ana', 'Ante', 'Boris', 'Maja', 'Marko',
           'Mirko', 'Slavko', 'Slavica' ];
$q = $_GET[ 'q' ];

foreach( $imena as $ime )
  if( strpos( $ime, $q ) !== false )
    echo '<option value="' . $ime . '">' . "\n";
?>
```

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`.



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.

Zadatak 2

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.

Ajax

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 );
```

Ajax - long polling

Long polling (Comet)

1. Klijent otvori zahtjev prema serveru.
2. Server ne odgovara na zahtjev sve dok nema spremne podatke:
   - 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.
3. Kada su podaci spremni, server ih pošalje klijentu i prekine vezu.
4. Klijent primi podatke i ponovno uspostavlja vezu sa serverom.

Mana ovog pristupa:

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

Primjer 3 - Chat

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).

Primjer 3 - Chat

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).
  - Svakih 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.)

Ajax - završne napomene

- Ranije je adresa skripte koju kontaktiramo preko Ajax-a morala biti na istoj domeni kao i HTML.
- Danas je to ograničenje prevaziđeno (vidi CORS).

- Long polling je dobro rješenje ako su poruke relativno rijetke. Ovo je dovoljno za nas.
- U protivnom, moguća rješenja su:
  - WebSocket - dvosmjerna komunikacija, za složene primjene poput online igara, real-time sustava i slično. Specijalni protokol.
  - Server-sent events (SSE) - jednostavnije od WebSocket-a, za jednosmjernu komunikaciju server → klijent. HTTP protokol.