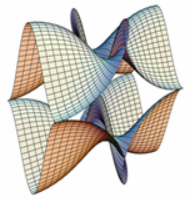


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Predavanje 11 - Asinkroni JavaScript.

9. siječnja 2026.

Sastavio: Zvonimir Bujanović

Ajax

Ajax - Asynchronous JavaScript + XML (povijesni naziv)

- 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, nego i tekst/JSON/HTML/binarni podaci).
- Danas se najčešće koristi u kombinaciji sa tzv. JSON-om.
- U praksi se Ajax najčešće koristi za pozive prema tzv. REST API-ju.

- Mi ćemo kontaktirati server pomoću modernog Fetch API-ja.
- Alternative:
  - XMLHttpRequest – starija i nespretna verzija.
  - $.ajax() – jednostavno korištenje Ajax-a pomoću jQuery.
  - Axios – popularna zamjena za Fetch API.

JSON - JavaScript Object Notation

- Služi za reprezentaciju JavaScript objekata u obliku stringa.
  - 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:

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

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

```
1 document.querySelector('#p2').innerHTML +=
2   '<br>JMBAG: ' + s[0].JMBAG + ' ime: ' + s[0].ime +
3   '<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 na serverskoj strani (Flask)

- U Pythonu, uz `import json`:
  - `let str = JSON.stringify(obj)` ⇝ `str = json.dumps(obj);`
  - `let obj = JSON.parse(str)` ⇝ `obj = json.loads(str);`

- U nastavku ćemo htjeti da Flask generira JSON umjesto HTML-a!
  - Do sada: ruta u Flasku vraća string ili `render_template()` ⇝ generira se *response* kao HTTP poruka koja u zaglavlju (*headeru*) sadrži `'Content-type:text/html;charset=utf-8'`
  - Sada: želimo *response* koji u zaglavlju ima `'Content-type:application/json;charset=utf-8'`

To se postiže pomoću Flaskove funkcije `jsonify()`:

```
1 from flask import Flask, jsonify;
2 ...
3
4 @app.route("/data")
5 def data():
6     # Pretvori dict u JSON string i dodatno generiraj HTTP poruku sa
7     # zaglavljem 'Content-type:application/json;charset=utf-8'
8     return jsonify( { "message": "Hello", "items": [1, 2, 3] } );
```

Asinkroni JavaScript

Dohvaćanje podataka sa servera nije trenutno.

- Takav kod koji se izvršava dulje (komunikacija preko mreže, učitavanje s diska, čekanje na timeout, ...) ne smije "zamrznuti" web-stranicu.
- Zato se takve operacije rade asinkrono: browser odrađuje posao u pozadini, te JavaScriptu kasnije dojavi rezultat.

JavaScript koristi mehanizam tzv. Promise-a za *dojavljivanje kasnije*.

- Bazična sintaksa (`Promise`, `resolve`, `then`) je relativno komplicirana.
- Novija sintaksa, puno jednostavnije: `async`, `await`.
  - Sažetak u jednoj rečenici: `await` pauzira samo `async` funkciju u kojoj se nalazi kako bi pričekao da mu `Promise` dojavi rezultat, a ostatak programa može nastaviti izvršavanje.

Promise - Osnove

Promise

- Ako se nešto izvršava asinkrono (mreža/IO/timeout), *promise* omogućava da se neki kod izvrši po završetku.
- Greške možemo detektirati članskom funkcijom catch.
- Ispis donjeg koda je: `"prije poslije"` i onda za 5 sekundi `"Uspješno riješen promise: gotovo"`.

```
1  let promise = new Promise( function(resolve, reject) {
2      // Ova (anonimna) funkcija se odmah počne izvršavati.
3      // setTimeout je asinkron i izlazimo iz anonimne funkcije.
4      // Za 5 sekundi se "promise" završi s rezultatom "gotovo".
5      setTimeout( function() { resolve( 'gotovo' ); }, 5000 );
6  });
7
8  console.log( 'prije' );
9  promise.then( function(result)
10 {
11     // Kod koji se izvrši kada se "promise" uspješno završi.
12     console.log( 'Uspješno riješen promise: ' + result );
13 } );
14 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      // await pauzira funkciju f na idućoj liniji dok se promise ne završi.
7      // JavaScript nije blokiran i izvođenje koda izvan funkcije f se nastavlja.
8      let result = await promise;
9      console.log( result );
10 }
11
12 console.log( 'prije' ); f(); console.log( 'poslije' );
```

Ajax - Klijentska strana: GET

Fetch API

- Funkcija **`fetch`** asinkrono dohvaća podatke sa zadanog URL-a.
- Vraća *promise* čijim uspješnim završetkom dobivamo podatke.
- Donji primjer šalje Flask aplikaciji 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?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();
```

Ajax - Serverska strana: GET

Flask:

- Klijent je poslao podatke putem GET-a.
- Dohvaćamo ih pomoću `request.args`.
- Generiramo *response* u obliku JSON-a.

```
1  from flask import request, jsonify;
2
3  @app.route( '/getinfo' )
4  def getinfo():
5      user = request.args.get('user');
6      age = request.args.get('age', type=int);
7
8      # Sad tu radimo nešto sa user, age...
9
10     return jsonify( {'status': 'ok'} );
```

Ajax - Klijentska strana: POST, šaljemo JSON

Fetch API

- Ako serveru šaljemo podatke pomoću POST ⇝ opcije.
- Običaj je i serveru slati podatke u JSON formatu.

```
async function post_data()
{
    // Ajax poziv sa POST: parametri se šalju kroz opciju body.
    let response = await fetch( '/postinfo', {
        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();
```

Ajax - Serverska strana: POST, primamo JSON

Flask:

- Klijent je poslao podatke putem POST-a u JSON formatu.
- Dohvaćamo ih pomoću `request.get_json();`
- Generiramo *response* u obliku JSON-a.

```
1  from flask import request, jsonify;
2
3  @app.route( '/postinfo', methods=['POST'] )
4  def postinfo():
5      data = request.get_json();
6
7      user = data.get('user'); # ili: user = data['user'];
8      age = data.get('age');   # ili: age = data['age'];
9
10     # Sad tu radimo nešto sa user, age...
11
12     return jsonify( {'status': 'ok'} );
```

Ajax - Klijentska strana: POST, šaljemo FormData

Fetch API

- Ako serveru šaljemo podatke pomoću POST ⇝ opcije.
- Moguće je simulirati i slanje forme. To se obično ne radi, osim ako ne uploadamo neku datoteku.

```
1  async function post_data()
2  {
3      let fd = new FormData();
4      fd.append( 'user', 'Ana' );
5      fd.append( 'age', 21 );
6      fd.append( 'file', input.files[0] );
7
8      // Ajax poziv sa POST: parametri se šalju kroz opciju body.
9      let response = await fetch( '/postinfo', {
10         method: 'POST',
11         body: fd
12     } );
13
14     // Dohvatimo podatke iz response-a.
15     let data = await response.text(); // ili .json()
16 }
17
18 post_data();
```

Ajax - Serverska strana: POST, primamo FormData

Flask:

- Klijent je poslao podatke putem POST-a kroz FormData.
- Efekt je kao da je popunio formu i poslao ju preko POST.
- Generiramo *response* u obliku JSON-a.

```
1  from flask import request, jsonify;
2
3  @app.route( '/postinfo', methods=['POST'] )
4  def postinfo():
5      user = request.form.get('user');
6      age = request.form.get('age', type=int);
7      file = request.files.get('file');
8
9      # Sad tu radimo nešto sa user, age, file...
10
11     return jsonify( {'status': 'ok'} );
```

Ajax - Fetch API - Detekcija grešaka

Fetch API

- Greške hvatamo s `try/catch` (kod async/await) ili s `.catch` (kod Promise-a).
- `fetch` baca iznimku za mrežne greške.
- Za neke HTTP odgovore (status kodovi 4xx/5xx) `fetch` ne baca iznimku ⇝ treba provjeriti `response.ok`.

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

    if( !response.ok ) {
        throw new Error( `HTTP ${response.status}` );
    }

    // Pričekamo JSON i onda ga vratimo.
    // Uoči, async funkcija prije no što se to dogodi vrati Promise!
    return await response.json();
}

loadJson( '/getinfo' )
    .then( json => console.log(json) )
    .catch( err => alert(err.message) );
```

jQuery - Ajax

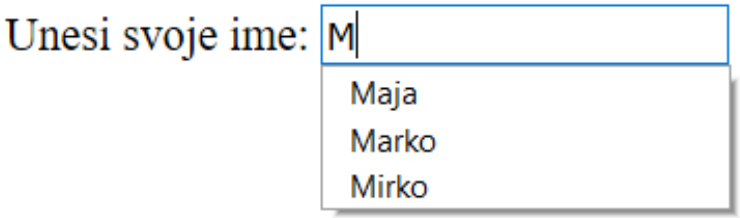
Ako koristimo jQuery, u njemu je dostupna funkcija .ajax:

```
1  $.ajax(
2  {
3      url: '/getinfo',
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 – `primjer2.html`

```
txt.addEventListener( 'input', async function() {
    let unos = this.value;

    // Napravi Ajax poziv sa GET i dobij sva imena koja sadrže s kao podstring.
    let response = await fetch( `/suggest?q=${unos}` );
    let list = await response.json(); // Flask vraća listu u JSON formatu.

    let datalist = document.querySelector( '#datalist_imena' );

    // Prvo isprazni listu od prethodnih opcija.
    datalist.innerHTML = '';
    for( let ime of list )
        datalist.innerHTML += `<option value="${ime}">`;
} );
```

Primjer 2 - "Suggest"

Serverski dio – `app.py`

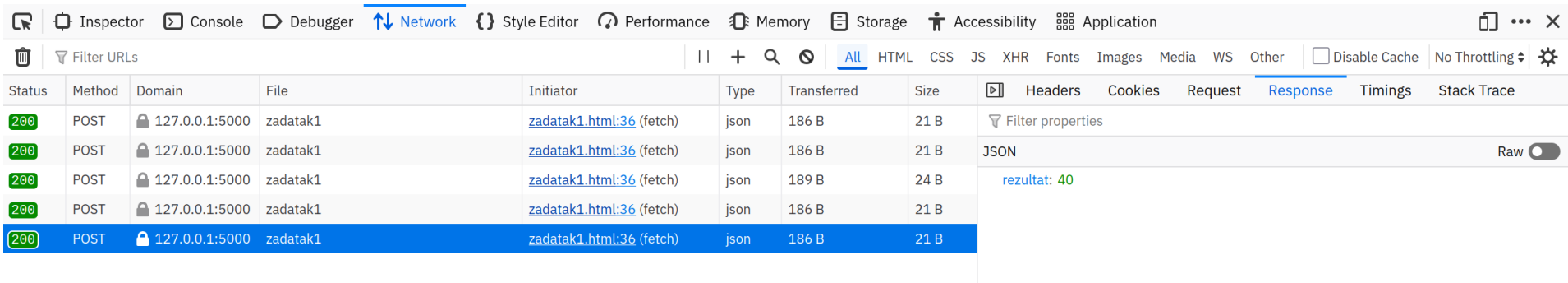
```
@app.route( '/suggest' )
def suggest():
    imena = [ 'Ana', 'Ante', 'Boris', 'Maja', 'Marko', 'Mirko' ];

    # Dohvati vrijednost pridruženu ključu q poslanu putem GET-a.
    q = request.args.get('q', '');

    # Protrči kroz sva imena i vrati HTML kod <option> za samo ona
    # koja sadrže string q kao podstring.
    lista = [];
    for ime in imena:
        if( q in ime ): # Je li q podstring od ime?
            lista.append( ime );

    # Vrati listu s imenima u JSON formatu.
    return jsonify( lista );
```

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 Pythonu).
  - Server treba generirati detaljne poruke o greškama.
  - Korištenje Web Developer alata u browseru (tab Network, Response).
  - Direktan pristup serverskoj ruti u browseru, npr. `/suggest?q=M`.
  - Korištenje alata iz komandne linije:

```
curl -X POST "http://127.0.0.1:5000/getinfo"
     -H "Content-Type: application/json"
     -d '{"user": "Ana", "age": 21}'
```

  - Trik: tab Network ⇝ Copy as cURL.



Zadatak 1

Napravite jednostavni kalkulator (4 osnovne računske operacije).

- Klijent šalje operande i operator, a 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.
- Pogledajte Pythonove funkcije urlparse i pathlib.stat().

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 s rute `/cekajPoruku`.
  - 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 ruta `/posaljiPoruku`.
  - 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 (Flask):

- Ruta `/posaljiPoruku`:
  - Iz GET-a pročita stringove `ime` (ime korisnika koji šalje poruku) i `msg` (sadržaj poruke).
  - Zapiše te stringove u datoteku `chat.log`.
- Ruta `/cekajPoruku`:
  - Iz GET-a pročita `timestamp` (vrijeme kad je zadnji put primljena poruka), `cache` (trenutno vrijeme).
  - Svakih 0.5s 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 rwa-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 Flask rutu `/zadatak3`.
  - Ta ruta neka vraća podatke u JSON formatu (polje).
  - Server neka u petlji "odspava" bar 1s prije novog upita na bazu!
  - 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.) Važno: Prije SELECT-a treba napraviti `db.commit();`

Ajax - završne napomene

- Po defaultu, adresa serverske skripte koju kontaktiramo preko Ajax-a mora biti na istoj domeni kao i HTML (tzv. *same-origin*).
- Server može eksplicitno dozvoliti pristup vanjskim zahtjevima (CORS).

- Long polling je prihvatljivo rješenje ako su poruke relativno rijetke. Ovo je dovoljno za nas.
- U protivnom, moguća moderna 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 $\rightarrow$ klijent. HTTP protokol.