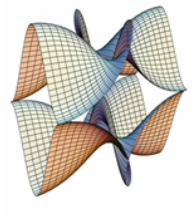


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM II

Predavanje 10 - Web API za JavaScript

28. svibnja 2018.

Sastavio: Zvonimir Bujanović

Web API za JavaScript

Document Object Model i Browser Object Model daju dva osnovna API-ja u JavaScriptu. Postoji i velik broj drugih.

U ovom predavanju ćemo dati kratki pregled nekih Web API-ja koji dolaze ugrađeni u aktualnu generaciju browsera:

- Geolocation
- Canvas
- Web Worker
- WebSocket
- Storage
- IndexedDB

Demonstraciju mogućnosti raznih Web API-ja možete vidjeti ovdje.

Geolocation

- Omogućava dohvaćanje geografske lokacije korisnika (ako to korisnik dozvoljava).
- Jedan od najjednostavnijih API-ja:
  - `navigator.geolocation.getCurrentPosition(suc, err)`
    Otkriva trenutnu lokaciju korisnika. Poziva funkciju `suc` ako ju uspije doznati, a (opcionalnu) `err` ako ne.
  - `id = navigator.geolocation.watchPosition(suc, err)`
    Prati lokaciju korisnika. Svaki put kad se ona primijeni, poziva funkciju `suc`, u protivnom (opcionalnu) `err`.
  - `navigator.geolocation.clearWatch(id)`
    Otkazuje daljnje praćenje lokacije korisnika.

Primjer 1 - Geolocation (i Google Maps)

Na klik gumba se:

- Određuje trenutna lokacija korisnika.
- Pomoću Google Maps API se prikazuje mapa okolnog područja, te marker na lokaciji korisnika.

U primjeru koristimo najjednostavniju, statičku mapu dohvaćenu kao slika.

- Puna dokumentacija za Google Maps Web API.
- Tutorial za Google Maps API na w3schools.

Canvas

- HTML5 API koji omogućava potpunu programsku kontrolu nad kreiranjem grafičkih elemenata.
- Unutar HTML dokumenta kreira se (prazni) element tipa `canvas` odgovarajuće širine i visine:

```
<canvas height="500" width="500" id="cnv">
    Ovaj tekst se prikaže ako canvas nije podržan.
</canvas>
```

- Nakon toga, svi grafički elementi se dodaju pomoću JavaScripta, te crtaju u odgovarajućem "kontekstu":

```
var canvas = $( "#cnv" ).get(0);
var ctx = canvas.getContext( "2d" );
```

- Za 2D-grafiku koristimo grafički kontekst `"2d"`.
- Za 3D-grafiku koristimo grafički kontekst `"WebGL"`.

Canvas - Osnovno korištenje 2d-konteksta

Kad dohvatimo 2d-kontekst, možemo crtati koristeći brojne funkcije:

- `ctx.lineWidth` – debljina linije kojom crtamo
- `ctx.strokeStyle` – definira boju kojom crtamo, npr. `"red"` ili `"rgb(10, 20, 255, 0.5)"`.
- `ctx.fillStyle` – definira boju kojom ispunjavamo plohe, npr. `"#0000ff"`
- `ctx.strokeRect(x1, y1, w, h)` – crta rub pravokutnika
- `ctx.fillRect(x1, y1, w, h)` – ispunjava pravokutnik
- Crtanje krivulja:
  - `ctx.beginPath()` – početak crtanja krivulje (“puta”)
  - `ctx.moveTo(x,y)` – pomiče se u točku s koordinatama (x, y)
  - `ctx.lineTo(x,y)` – crta liniju od kraja preth. dijela puta do (x, y)
  - `ctx.arc(cx, cy, r, phi0, phi1, ccw)` – crta kružni isječak
  - `ctx.bezierCurveTo(...)`, `ctx.quadraticCurveTo(...)`
  - `ctx.stroke()` – kraj crtanja krivulje
  - `ctx.fill()` – ispunjava zatvorenu krivilju bojom

Canvas - Osnovno korištenje 2d-konteksta

Neka svojstva i funkcije za pisanje teksta:

- `ctx.font` – font kojim pišemo, npr. `"bold 14px Arial"`
- `ctx.textAlign` – `"left"`, `"right"`, `"center"`
- `ctx.strokeText(tekst, x, y)` – piše tekst na koordinate (x, y) koristeći ctx.strokeStyle
- `ctx.fillText(tekst, x, y)` – piše tekst na koordinate (x, y) koristeći ctx.fillStyle

Prikazivanje slika iz datoteka:

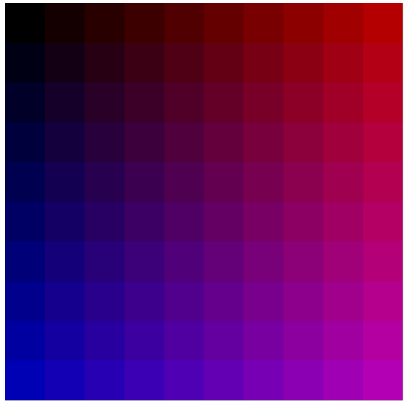
- `ctx.drawImage(img, x, y, w, h)` – crta sliku `img` (tipa `Image`) tako da joj je gornji lijevi kut u (x, y), te da je široka `w`, a visoka `h` pixela.

```
1 var img = new Image();
2 img.src = "slika.png";
3 $( img ).on( "load", function() {
4     ctx.drawImage( this, 50, 50, 100, 100 );
5 } );
```

Primjer 2 - Osnovno crtanje po Canvas-u

Primjer prikazuje:

- Kako nacrtati krivulju.
- Kako odabrati boju popunjavanja i nacrtati popunjeni pravokutnik.
- Kako pisati tekst po canvas-u.
  Uoči: tekst se ne može selektirati mišem, tj. zapravo je slika!



Hello

Napomena:

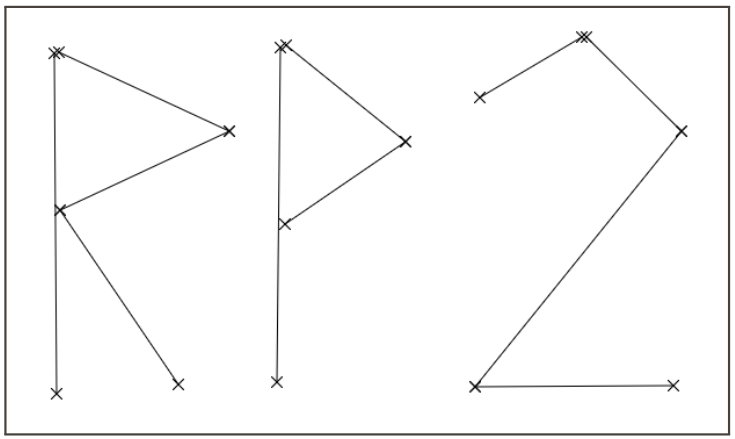
- Nakon selekcije canvas-a pomoću jQuery, moramo ga konvertirati iz jQuery objekta u "obični" HTML objekt.
- Ako je `j` neka jQuery kolekcija, onda pomoću
  - `j.at(i)` dobivamo i-ti objekt iz kolekcije kao jQuery objekt;
  - `j.get(i)` dobivamo i-ti objekt iz kolekcije kao HTML objekt.

Zadatak 1

Napravite JavaScript program za crtanje dužina:

- Prvi klik na canvas označi sa X početak dužine.
- Drugi klik na canvas označi sa X kraj dužine i spoji ju s početkom.
- Ovaj postupak se ponavlja.
- Uoči: `event.clientX` vraća koordinate klika unutar dokumenta. Zato treba oduzeti koordinate gornjeg lijevog vrha pravokutnika HTML elementa canvas:

```
1 var rect = canvas.getBoundingClientRect();
2 var x = event.clientX - rect.left,
3     y = event.clientY - rect.top;
```



Canvas - Osnovno korištenje 2d-konteksta

Transformacije koordinata:

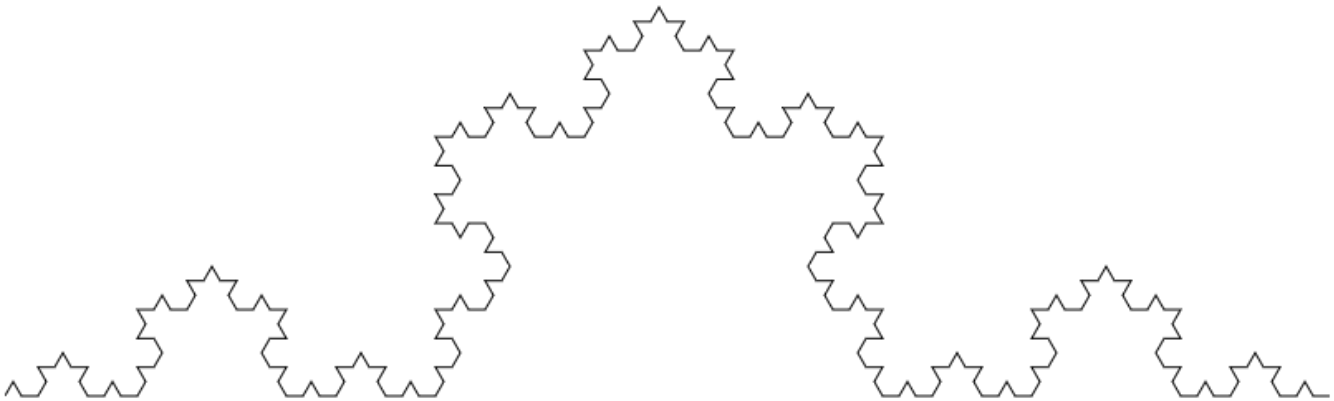
- `ctx.rotate(phi)` – od sada nadalje sve se crta zarotirano za kut phi oko ishodišta (u smjeru kazaljke na satu).
- `ctx.scale(dx, dy)` – od sada nadalje sve se crta skalirano za faktor (dx, dy).
- `ctx.translate(dx, dy)` – od sada nadalje sve se crta translatirano za (dx, dy), tj. ishodište se pomiče u točku (dx, dy).
- `ctx.save()` – zapamti trenutnu matricu transformacije (na stogu).
- `ctx.restore()` – vrati zadnje zapamćenu matricu transformacije (s vrha stoga).

Kompozicija svih primijenjenih transformacija se akumulira u matricu transformacije.

Canvas sadrži još brojne druge efekte, npr. gradijentne prijelaze, sjene, uzorke.

Primjer 3 - Transformacija koordinata (Kochova pahulja)

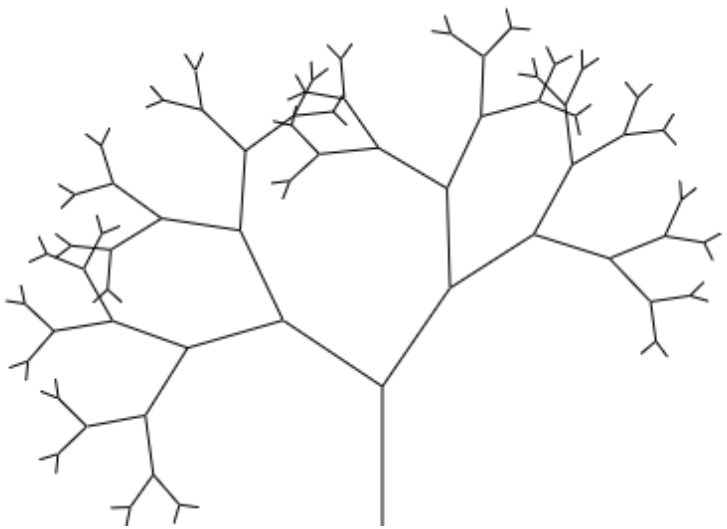
Korištenjem transformacija koordinata možemo crtati vrlo slično kao u LOGO-u ("turtle graphics").



Zadatak 2 (DZ) - Rekurzivno crtanje stabla

Korištenjem transformacija koordinata nacrtajte binarno stablo.

Prvo napravite simetrično stablo s granama pod kutem od $\pi/4$, a zatim (kao na slici) sa slučajno odabranim kutem kojeg zatvaraju grane s deblom.



Primjer 4 - Animacija analognog sata

Korištenjem transformacija koordinata i funkcije `requestAnimationFrame` možemo napraviti analogni sat.

Primjer je preuzet sa MDN.

Web Storage API

- Daje mehanizam kojem browseri mogu pohranjivati parove (ključ, vrijednost) na klijentskom računalu.
- Efikasna i jednostavna zamjena za *cookie*.
- Tipično se može čuvati bar 5MB podataka po domeni.
- Postoje dva mehanizma (svojstva objekta `window`):
  - `sessionStorage` – podaci se čuvaju tijekom jedne korisnikove sesije (dok ne zatvori stranicu)
  - `localStorage` – podaci se čuvaju i nakon zatvaranja browsera
- Klasa Storage:
  - `storage.setItem( key, value )` – dodavanje novog para u spremište (oba parametra su string).
  - `value = storage.getItem( key )` – vraća vrijednost za zadani ključ iz spremišta (ili `null` ako ne postoji).
  - `storage.removeItem( key )` – uklanjanje para sa zadanim ključem iz spremišta.
  - `storage.clear()` – brisanje svih ključeva iz spremišta.

Primjer 5 - WebStorage API

Primjer pokazuje sljedeće:

- Podaci spremljeni u localStorage (datum zadnje posjete) ostaju dostupni i nakon zatvaranja i ponovnog učitavanja stranice.
- Podaci spremljeni u sessionStorage (boja pozadine) se izgubi nakon zatvaranja stranice.

Druge moguće primjene WebStorage API:

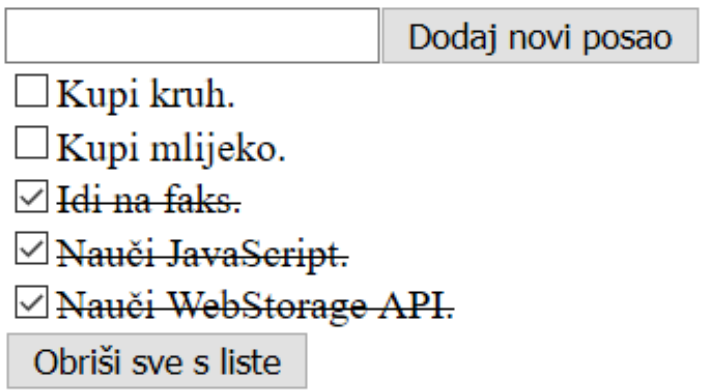
- Lokalni spremnik (cache) podataka sa servera ⇝ idući posjet web-stranici će biti puno brži.
- Spremanje postavki sučelja web-aplikacije prema korisnikovom izboru (npr. raspored menija, font, popis zadnje otvorenih datoteka i slično).

Podatke spremljene u localStorage možemo vidjeti u Storage Inspectoru u Firefoxu: u Debuggeru klik na Toolbox Options -> Default Firefox Developer Tools -> Storage.

Zadatak 3

Napišite JavaScript program za održavanje todo-liste poslova:

- Korisnik može dodati novi posao na kraj liste.
- (DZ) Korisnik može kvačicom (checkbox) označiti neki posao kao obavljen. Tada taj posao treba prekrižiti (ali ne i obrisati s liste).
- Korisnik može klikom na gumb obrisati cijelu listu poslova.
- Popis poslova treba biti dostupan i prilikom idućeg posjeta stranici.



Web Worker

- Daje jednostavan način za izvođenje neke skripte u pozadini, bez utjecaja na performanse web-stranice.
- Inicijalno, sav JavaScript kod se odvija u jednoj dretvi (thread-u) i stoga je nužno sekvencijalan.
- Web Worker-i omogućavaju stvaranja novih dretvi koje se odvijaju paralelno s inicijalnom dretvom.
- Nakon kreiranja,
  - Worker se izvršava u drugom globalnom kontekstu, različitom od `window`;
  - Worker zato ne može direktno pristupiti npr. DOM-u niti jQuery;
  - Worker može direktno pristupiti samo nekim API-jima, vidi listu;
  - Worker i inicijalna dretva mogu komunicirati slanjem poruka i odgovarajućih *event handlera*.

Dedicated Worker

- Kreiranje “radnika” iz inicijalne dretve:

```
var myWorker = new Worker( "worker.js" );
```

- Inicijalna dretva može slati poruku “radniku” ovako:

```
myWorker.postMessage( msg );
```

Ovdje je `msg` bilo kakva JavaScript varijabla.

- Worker reagira na poruku iz inicijalne dretve ovako (`worker.js`):

```
onmessage = function(e) { console.log( e.data ); }
```

- Worker može slati poruku inicijalnoj dretvi ovako (`worker.js`):

```
postMessage( msg );
```

- Inicijalna dretva reagira na poruku Worker-a ovako:

```
myWorker.onmessage = function(e) { console.log( e.data ); }
```

Dedicated Worker

- Inicijalna dretva može uništiti Worker-a ovako:

```
1 myWorker.terminate();
```

- Worker se sam može zaustaviti sa (`worker.js`):

```
1 close();
```

- Worker-i mogu i sami stvarati nove “podradnike”.
- Postoje i druge vrste Workera, na primjer SharedWorker kojeg može istovremeno dijeliti nekoliko web-stranica.

Primjer 6 - Računanje u pozadini

Računamo prvih 50 Fibonaccijevih brojeva (loše - rekurzivno).

- Računanje bez workera potpuno zaglavi web stranicu. Ona ne reagira na klikove na button.
- Računanje u pozadini pomoću workera omogućava da stranica posve normalno funkcionira i reagira na klikove na gumb.

  Pojedini broj se računa ovako:
  - Glavna dretva pošalje poruku workeru s zahjevom za izračun n-tog Fibonaccijevog broja;
  - Worker računa u pozadini, a glavna dretva neometano nastavlja raditi;
  - Kad worker završi izračun, on pošalje poruku glavnoj dretvi;
  - Glavna dretva reagira na poruku tako da doda novi span s izračunatim brojem u dokument.

Zadatak 4

Nadopunite rješenje Zadatka 1 tako da:

- Napravite 4 identična workera.
- Svakom workeru prvo pošaljete njegovo ime.
- Svaki worker "odspava" neki random broj sekundi, a zatim pošalje glavnoj dretvi random koordinate na canvasu, random kut, te svoje ime. Ovu proceduru neprestano ponavlja.
- Glavna dretva po primitku poruke treba napisati to ime na poslanim koordinatama, zarotirano za poslani kut.

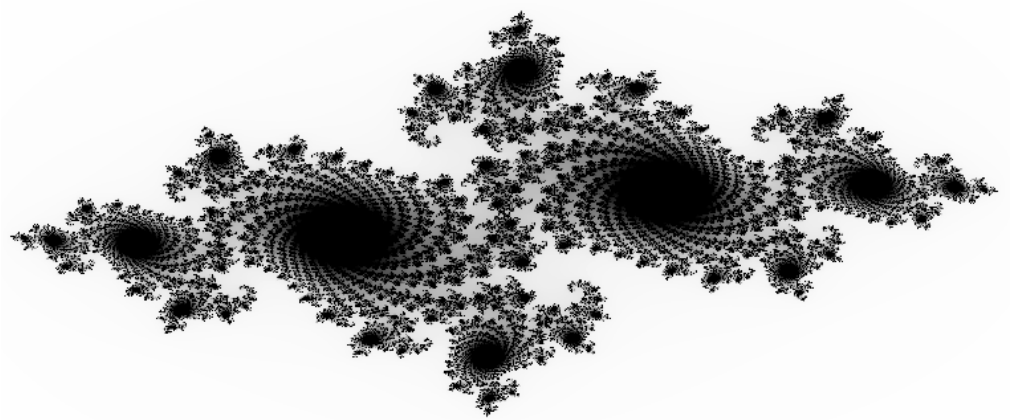
Napomene:

- Workeri mogu koristiti funkcije `setInterval`, `setTimeout`.
- Debugiranje u FireFox-u ⇝ u Debuggeru je na desnoj strani popis svih workera ⇝ dvoklik.

Primjer 7 - Julia set

Još jedan primjer kako izračunavanje u pozadini može:

- Ubrzati aplikaciju zbog dodatnih dretvi koji koriste više jezgri procesora.
- Učiniti aplikaciju responzivnijom zbog toga što glavna dretva nije zaglavljena računanjem.



WebSocket

- Protokol koji omogućava komunikaciju klijenta i servera pomoću jedne TCP konekcije.
- Prednosti u odnosu na HTTP:
  - WebSocket omogućava full-duplex komunikaciju.
  - WebSocket je efikasniji jer koristi puno manji header u odnosu na HTTP.
- "Životni ciklus" WebSocket-a:
  1. Klijent (browser) šalje serveru zahtjev za spajanje ("handshake") u obliku HTTP headera sa specijalnim "upgrade" poljem.
  2. Server odgovara šaljući drugi HTTP header.
  3. Time je uspostavljena konekcija te se komunikacija nastavlja po WebSocket protokolu.
  4. Klijent i server mogu razmjenjivati poruke u bilo kojem smjeru sve dok se veza ne zatvori.
- Za serversku stranu koristi se npr. socket.io.
  Apache nije pogodan za WebSocket-e.

WebSocket

- Otvaranje konekcije sa serverom
  (podržane URL sheme su `ws:` i `wss:`)

```
1 var socket = new WebSocket( url );
```

- Konekcija može reagirati na događaje pridruživanjem metoda:
  - `onopen` – aktivira se po ostvarenju konekcije
  - `onerror` – aktivira se po detekciji greške
  - `onmessage` – aktivira se po primitku poruke od strane servera
  - `onclose` – aktivira se po primitku poruke od strane servera
- Slanje podataka serveru:

```
1 socket.send( string );
```

  Moguće je slati i binarne podatke u obliku `ArrayBuffer` ili `Blob`.
- Zatvaranje konekcije sa serverom:

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

Primjer 8 - WebSocket

U primjeru se:

- Spajamo na WebSocket server na adresi `ws://echo.websocket.org/`.
- Riječ je o echo serveru, pa što god mu pošaljemo, automatski nam vrati natrag.

IndexedDB

- Za razliku od Web Storage, služi za čuvanje veće količine strukturiranih podataka na strani klijenta.
- Omogućuje brzo pretraživanje ovih podataka pomoću indeksa.
- Objektno-orijentirana baza podataka u JavaScriptu.
- Operacije iz IndexedDB se odvijaju asinkrono (bez da koristimo Worker!).
- Max. količina podataka ovisi o browseru (Firefox nema limit).
- Tipični scenario:
  1. Otvori bazu podataka.
  2. Kreiraj "object store" u bazi podataka.
  3. Pokreni zahtjev za operacijom na bazi, poput dodavanja ili dohvaćanja podataka.
  4. Čekaj da se operacija završi osluškivanjem odgovarajućeg događaja.
  5. Napravi nešto s dohvaćenim rezultatima.
- Tipična namjena: offline aplikacije.

Primjer 9 - IndexedDB (Stvaranje nove baze podataka)

```
var request = indexedDB.open( "pmf-mo", 1 ), db;

request.onupgradeneeded = function(e) {
  db = e.target.result;

  // Kreiramo objectStore koji će držati info o studentima.
  // "jmbag" je ključ jer jedinstveno određuje studenta.
  var objectStore = db.createObjectStore(
    "studenti", { keyPath: "jmbag" } );

  // Kreiramo index za pretraživanje po imenu.
  // Vise studenata moze imati isto ime => unique: false
  objectStore.createIndex( "ime", "ime", { unique: false } );

  // Kreiramo index za pretraživanje po emailu.
  // email je jedinstven za studenta => unique: true
  objectStore.createIndex( "email", "email", { unique: true } );
};
```

Primjer 9 - IndexedDB (Dodavanje novih podataka u bazu)

```
$("#btn").on( "click", function() {
  // Dodavanje novog studenta u bazu
  var student = {
    jmbag   : $("#jmbag").val(),
    ime     : $("#ime").val(),
    starost : $("#starost").val(),
    email   : $("#email").val()
  };

  // Kreiraj "readwrite" transakciju za dodavanje studenta
  // U transakciji sudjeluje objectStore "studenti".
  var transaction = db.transaction( ["studenti"], "readwrite" );

  // Dohvati objectStore iz transakcije
  var objectStore = transaction.objectStore( "studenti" );

  // Dodaj studenta u objectStore (opet, asinkrono)
  var request = objectStore.add( student );
}
```

Primjer 9 - IndexedDB (Dohvaćanje podataka iz baze)

```
$("#btnNadji").on( "click", function() {
  var jmbag = $("#txtNadji").val();

  // Započni transakciju na objectStoreu studenti
  var transaction = db.transaction( ["studenti"] );
  var objectStore = transaction.objectStore( "studenti" );

  // Zatraži element u objectStoreu s odgovarajućim ključem
  var request = objectStore.get( jmbag );

  request.onsuccess = function()
  {
    $("#spanNadji").html(
      request.result
        ? "E-mail je: " + request.result.email
        : "Nema takvog studenta" );
  };
}
```