



Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Predavanje 10 - Web API za JavaScript.

19. prosinca 2025.

Sastavio: Zvonimir Bujanović



Document Object Model i Browser Object Model daju dva osnovna Web API-ja u JavaScriptu. Moderni browseri nude 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 Storage (localStorage / sessionStorage)
- Web Worker
- WebSocket
- IndexedDB

Demonstraciju mogućnosti raznih Web API-ja možete vidjeti na <https://whatwebcando.today/> ili <https://webapis.co/>.

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.
- Geolocation API radi samo na HTTPS-u (ili na localhostu) – na HTTP stranicama neće raditi.

Na klik gumba se:

- Određuje trenutna lokacija korisnika.
- Pomoću biblioteke **OpenLayers** se prikazuje mapa okolnog područja.

U primjeru koristimo najjednostavniju, statičku mapu dohvaćenu kao slika.

- **Puna dokumentacija** za OpenLayers API.
- Napomena: Google Maps API zahtjeva registraciju i kreditnu karticu, iako je i dalje besplatan za mali broj prikazivanja mape.

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:

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

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

```
1 let canvas = document.querySelector( '#cnv' );  
2 let ctx = canvas.getContext( '2d' );
```

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

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

- `ctx.lineWidth` – debljina linije kojom crtamo
- `ctx.strokeStyle` – definira boju kojom crtamo, npr. "red" ili "rgba(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:

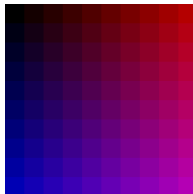
- `ctx.drawImage(img, x, y, w, h)` – crta sliku `img` (tipa `Image`) tako da joj je gornji lijevi kut u (x, y), te da je široka `w`, a visoka `h` pixela.

```
1 let img = new Image();
2 img.src = 'slika.png';
3 img.addEventListener( '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!



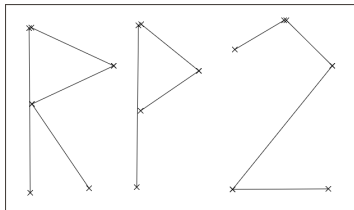
Hello

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



Transformacije koordinata:

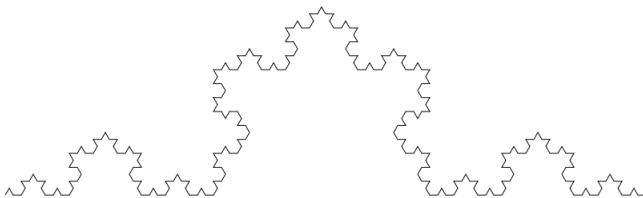
- `ctx.rotate(phi)` – od sada nadalje sve se crta zarotirano za kut ϕ oko ishodišta (u smjeru kazaljke na satu).
- `ctx.scale(dx, dy)` – od sada nadalje sve se crta skalirano za faktor (dx, dy).
- `ctx.translate(dx, dy)` – od sada nadalje sve se crta translaticirano za (dx, dy), tj. ishodište se pomiče 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)

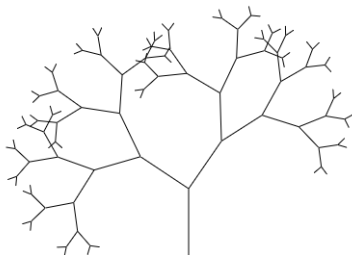
Korištenjem transformacija koordinata možemo crtati vrlo slično kao u LOGO-u ("turtle graphics").



Zadatak 2 (DZ) - Rekurzivno crtanje stabla

Korištenjem transformacija koordinata nacrtajte binarno stablo.

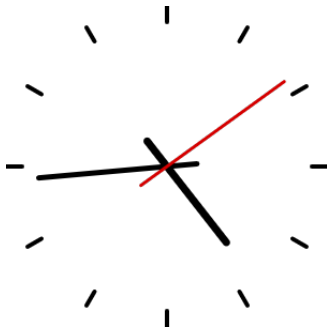
Prvo napravite simetrično stablo s granama pod kutem od $\pi/4$, a zatim (kao na slici) sa slučajno odabranim kutem kojeg zatvaraju grane s deblom.



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 alternativa za neke primjene *cookieja*.
- Tipično se može čuvati 5-10 MB podataka po domeni.
- Ne čuvati osjetljive podatke (lozinke, auth tokeni)!
- 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 $\rightsquigarrow$ 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).

Spremljene podatke možemo vidjeti u browseru pod Web developer tools -> Storage.

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.

Dodaj novi posao

☐ Kupi kruh.

☐ Kupi mlijeko.

☒ ~~Idi na faks.~~

☒ ~~Nauči JavaScript.~~

☒ ~~Nauči WebStorage API.~~

Obriši sve s liste

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 handler*a.
- Tipične primjene: parsiranje velikih datoteka, generiranje slika, složeni algoritmi, sve što bi inače *zamrznulo* UI.

Dedicated Worker

- Kreiranje “radnika” iz inicijalne dretve:

```
1 let wrk = new Worker( 'worker.js' );
```

- Inicijalna dretva može slati poruku “radniku” ovako:

```
1 wrk.postMessage( msg );
```

Ovdje je `msg` bilo kakva JavaScript varijabla.

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

```
1 self.addEventListener( 'message', function(e) { console.log( e.data ); } );
```

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

```
1 postMessage( msg );
```

- Inicijalna dretva reagira na poruku Worker-a ovako:

```
1 wrk.addEventListener( 'message', function(e) { console.log( e.data ); } );
```

Dedicated Worker

- Inicijalna dretva može uništiti Worker-a ovako:

```
1 wrk.terminate();
```

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

```
1 close();
```

- Worker-i mogu i sami stvarati nove “podradnike”.

Oprez

Worker se obično može pokrenuti samo korištenjem web-servera!

Browseri ne dozvoljavaju pokretanje `worker.js` ako je HTML učitao “dvoklikom” na datoteku.

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 zahtjevom 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.

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:

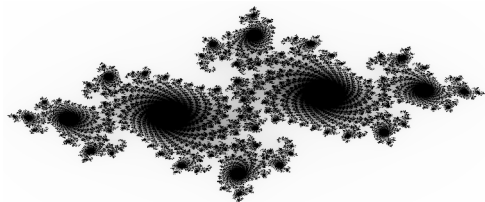
- Workeri mogu koristiti funkcije `setInterval`, `setTimeout`.
- Debugiranje u Firefox-u $\rightsquigarrow$ u Debuggeru je na desnoj strani popis svih workera $\rightsquigarrow$ dvoklik.

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.

U ovom primjeru crtamo jedan fraktal - tzv. **Julia set**.



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. biblioteka [socket.io](#). Za Python vidi i paket [websockets](#).

WebSocket

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

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

- Konekcija može reagirati na događaje (**addEventListener**):
 - **open** – aktivira se po ostvarenju konekcije
 - **error** – aktivira se po detekciji greške
 - **message** – aktivira se po primitku poruke od strane servera
 - **close** – 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();
```

U primjeru se:

- Spajamo na WebSocket server na adresi `wss://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 (stotine MB).
- 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)

```
1 let request = indexedDB.open( 'pmf-mo', 1 ), db;
2
3 request.addEventListener( 'upgradeneeded', function(e) {
4   db = e.target.result;
5
6   // Kreiramo objectStore koji će držati info o studentima.
7   // "jmbag" je ključ jer jedinstveno određuje studenta.
8   let objectStore = db.createObjectStore(
9     'studenti', { keyPath: 'jmbag' } );
10
11   // Kreiramo index za pretraživanje po imenu.
12   // Više studenata može imati isto ime => unique: false
13   objectStore.createIndex( 'ime', 'ime', { unique: false } );
14
15   // Kreiramo index za pretraživanje po emailu.
16   // email je jedinstven za studenta => unique: true
17   objectStore.createIndex( 'email', 'email', { unique: true } );
18 } );
```

Primjer 9 - IndexedDB (Dodavanje novih podataka u bazu)

```
1 document.querySelector('#btn').addEventListener( 'click', function() {  
2   // Dodavanje novog studenta u bazu  
3   let student = {  
4     jmbag   : document.querySelector('#jmbag').value,  
5     ime     : document.querySelector('#ime').value,  
6     starost : Number(document.querySelector('#starost').value),  
7     email   : document.querySelector('#email').value  
8   };  
9  
10  // Kreiraj "readwrite" transakciju za dodavanje studenta  
11  // U transakciji sudjeluje objectStore "studenti".  
12  let transaction = db.transaction( ['studenti'], 'readwrite' );  
13  
14  // Dohvati objectStore iz transakcije  
15  let objectStore = transaction.objectStore( 'studenti' );  
16  
17  // Dodaj studenta u objectStore (opet, asinkrono)  
18  let request = objectStore.add( student );  
19 } );
```

Primjer 9 - IndexedDB (Dohvaćanje podataka iz baze)

```
1 document.querySelector('#btnNadji').addEventListener( 'click', function() {
2   let jmbag = document.querySelector('#txtNadji').value;
3
4   // Započni transakciju na objectStoreu studenti
5   let transaction = db.transaction( ['studenti'], 'readonly' );
6   let objectStore = transaction.objectStore( 'studenti' );
7
8   // Zatraži element u objectStoreu s odgovarajućim ključem
9   let request = objectStore.get( jmbag );
10
11  request.addEventListener( 'success', function()
12  {
13    document.querySelector('#spanNadji').textContent =
14      request.result
15        ? 'E-mail je: ' + request.result.email
16        : 'Nema takvog studenta';
17  } );
18 }
```