

# **Napredni razvoj programske podpore za web**

**- predavanja -  
2021./2022.**

---

## **Napredni HTML(5)**

# Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.




*U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

# Uvod (1)

- Najvažnije pitanje za početak - da li netko ne zna HTML?
  - Molim dignite ruke... 
- HTML (*HyperText Markup Language*) je prezentacijski jezik za strukturiranje i prezentiranje sadržaja za web
  - Osnovni jezik koji upotrebljavamo za izradu web stranica. On web preglednicima daje podatke o sadržaju i strukturi učitane web stranice, a preglednik od tih podataka oblikuje i prikazuje stranicu
  - HTML je jezik za označavanje hipertekstualnih dokumenata
  - HTML je jezik za opis web stranica
  - Definirao ga je 1990. godine sir Timothy Berners-Lee, pri World Wide Web konzorciju (W3C), organizaciji koja brine o standardizaciji web tehnologija i razvoju weba

## Uvod (2)

- Najvažniji pojmovi za razumijevanje svrhe i primjene HTML-a, pa time i HTML-a 5:
  - HyperText
    - Tekstualna struktura (tekst) unutar nekog dokumenta koji sadržava poveznice na druge dokumente (npr. tekstualne)
    - „Hypertext is text which is not constrained to be linear.”
    - Hipertekst za razliku od tradicionalnog teksta nema jedinstven redoslijed čitanja, već ga korisnik (čitatelj) dinamički određuje
    - Namijenjen je prikazu na elektroničkom računalu, ne u tradicionalnim medijima (npr. knjiga)
  - Markup
    - Označavanje sadržaja u tekstualnoj strukturi

# Nova svojstva HTML5

- HTML5 je posljednji standard jezika HTML
  - Objavljen 01/2008., W3C Recommendation/update 10/2014.
- Što je novo u HTML5 u odnosu na prethodne verzije HTML-a?

# Provjera podrške u pregledniku

## ■ HTML5 *feature detection* biblioteka



- Detektira HTML5 značajke koje web preglednik podržava
- JavaScript, otvoreni kod, automatski se pokreće unutar HTML5 koda u web pregledniku koji se provjerava
- Dva načina rada:
  - Primjena sa CSS-om
  - Primjena sa JS-om

```
■ <head>  
■ <script src="modernizr.js"></script>  
■ </head>
```

```
■ .no-cssgradients .header {  
■   background: url("images/glossybutton.png");  
■ }  
■ .cssgradients .header {  
■   background-image: linear-gradient(cornflowerblue,  
■     rebeccapurple);  
■ }
```

# Dinamički HTML

- Statičko (static web) i dinamičko (dynamic web) upravljanje web sadržajem
- Manipulacija HTML dokumentom iz JavaScript programskog kôda
  - Dodavanje novih elemenata
  - Uklanjanje postojećih elementa
  - Izmjena položaja elemenata u DOM stablu
  - Izmjena sadržaja elementa (innerHTML)
  - Upravljanje CSS stilom elementa (formatiranje, vidljivost, položaj na ekranu, *layout*)
  - Obrada događaja

# Stablo Document Object Model (1)

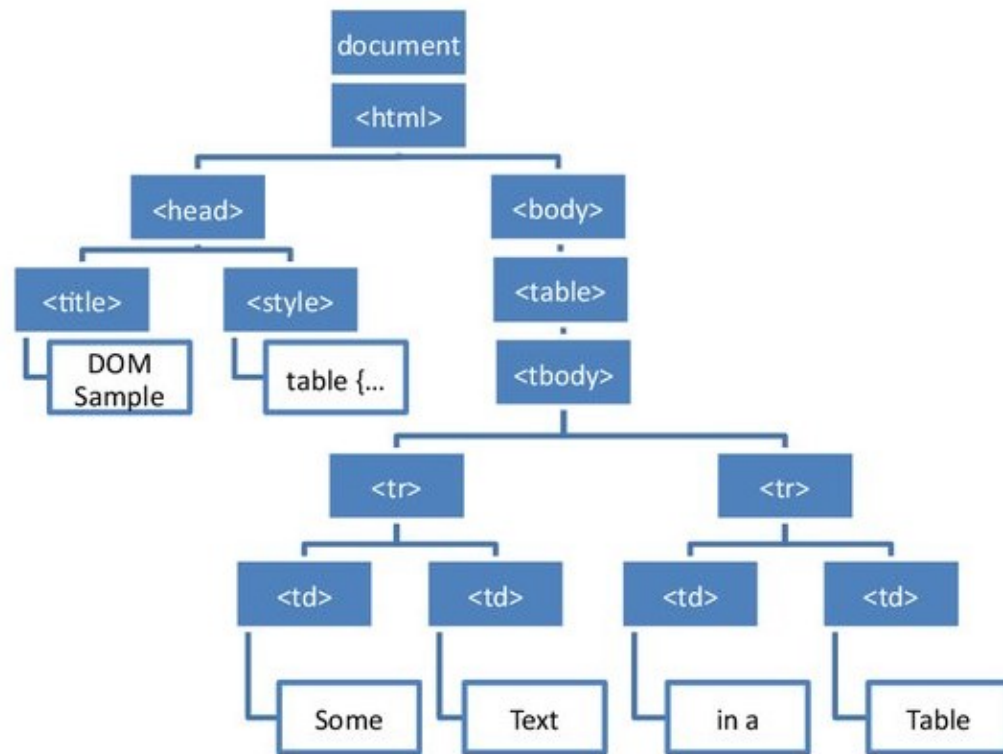
- DOM opisuje cjelovitu strukturu HTML5 dokumenta
  - Svi HTML elementi su objekti
  - Definira svojstva, metode i događaji svih HTML elemenata



```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Sample</title>
    <style type="text/css">
      table {
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <table>
      <tr>
        <td>Some Text in a Table</td>
      </tr>
    </table>
  </body>
</html>
```

## DOM Tree Traversal

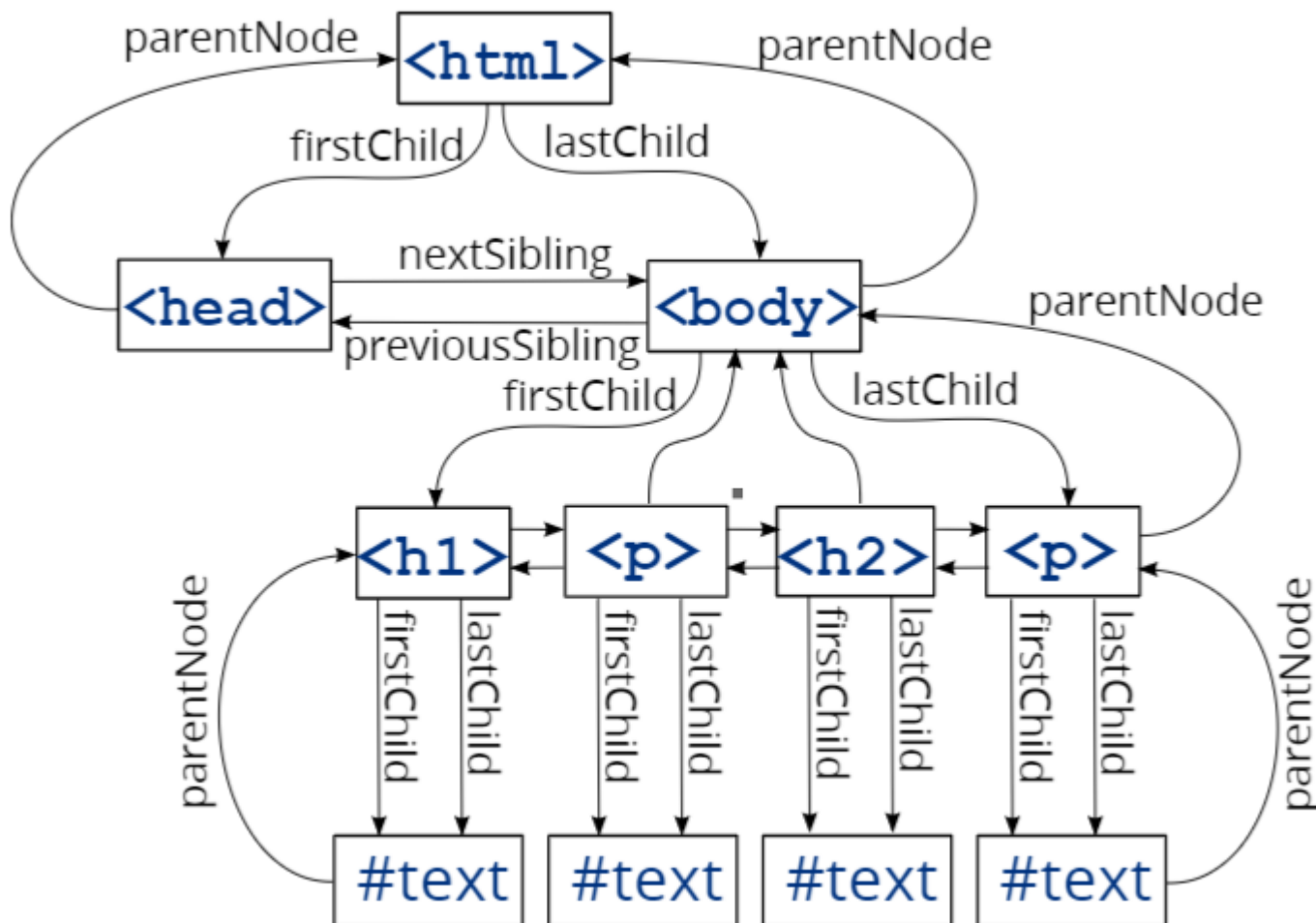
```
N.childNodes
N.firstChild
N.lastChild
N.nextSibling
N.ownerDocument
N.parentNode
N.previousSibling
```





# Stablo Document Object Model (2)

- Programski prolazak kroz DOM



# Dohvaćanje elemenata u DOM stablu (1)

- Dohvaćanje HTML5 elemenata moguće je na 4 načina:
  - Po jedinstvenoj šifri elementa (*by id*)
  - Po oznaci elementa (*by tag*)
  - Po nazivu razreda (klase) elementa (*by class*)

- `var t = document.getElementById('target');`

- `var t = document.getElementsByTagName('p');`

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p id="target">Sample Target</p>
    <p>Another Paragraph</p>
  </body>
</html>
```

# Dohvaćanje elemenata u DOM stablu (1)

- Dohvaćanje HTML5 elemenata moguće je na 4 načina:
  - Po jedinstvenoj šifri elementa (*by id*)
  - Po oznaci elementa (*by tag*)
  - Po nazivu razreda (klase) elementa (*by class*)

- `var t = document.getElementById('target');`

- `var t = document.getElementsByClassName('target');`

- `var t = document.getElementsByTagName('p');`

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p class="target">Sample Target</p>
    <p class="target">Another Paragraph</p>
  </body>
</html>
```

## Dohvaćanje elemenata u DOM stablu (2)

- Dohvaćanje pomoću CSS selektora u JavaScriptu
  - Funkcionalnost samo iz HTML5 API

```
document.querySelector("#section1");  
document.querySelectorAll("div");  
document.querySelectorAll(".section");
```

```
var childs = document.querySelectorAll('table tr td:first-child');
```

# Stvaranje novih elemenata (1)

- Novi elementi ne dodaju se u DOM stablo
  - Potrebno ih je dodati naknadno („ručno“)

```
document.createElement(tagName);  
document.createTextNode(text);  
  
// kloniranje elementa  
node.cloneNode();  
  
// kloniranje elementa i svih njegovih nasljednika (djeca)  
node.cloneNode(true);
```

## Stvaranje novih elemenata (2)

- Dodavanje novih elementa u DOM stablo

```
// element novi postaje lastChild od elementa node  
node.appendChild(novi);  
  
// dodaj novi u kolekciju djece elementa node prije elementa el  
node.insertBefore(novi, el);  
// ili ovako...  
node.insertBefore(novi, node.firstChild);  
  
// zamijeni stari element stari sa novim elementom novi  
node.replaceChild(novi, stari);  
// ili ovako...  
stari.parentNode.replaceChild(novi, stari);
```

# Brisanje elemenata

- Elemente je moguće trajno ukloniti iz strukture DOM stabla

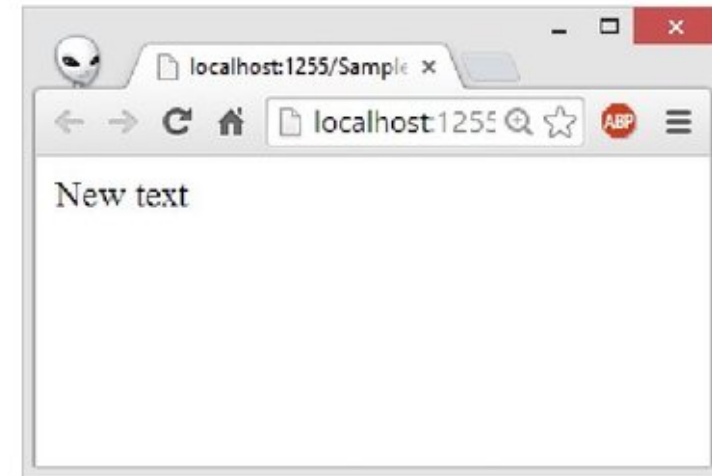
```
// izbriši element dijete od stari  
node.removeChild(stari);  
  
// izbriši element stari  
stari.parentNode.removeChild(stari);
```

# Izmjena elemenata u DOM stablu (1)

- Nakon dohvaćanja nekog HTML elementa moguće je izmijeniti:
  - HTML kôd elementa (*content*)
  - Vrijednost nekog atributa elementa (*attribute style*)
  - Formatiranje elementa (*style*)

```
document.getElementById(id).innerHTML = Nova vrijednost;
```

```
<html>
<body>
  <p id='target'>Old text</p>
  <script>
document.getElementById('target').innerHTML = "New
text";
  </script>
</body>
</html>
```

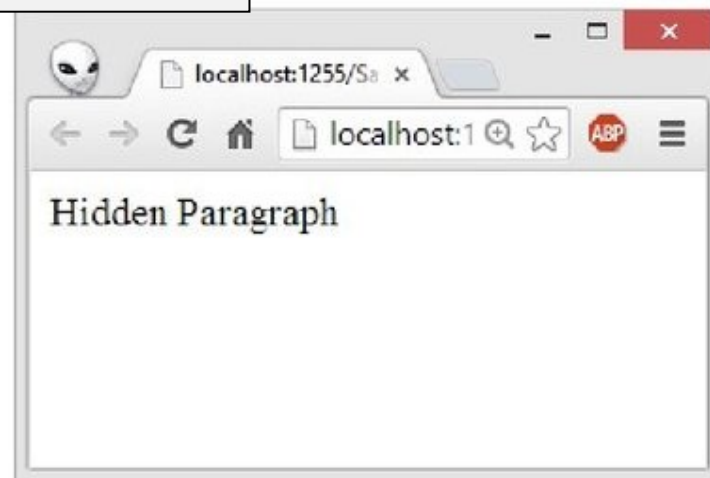




# Izmjena elemenata u DOM stablu (2)

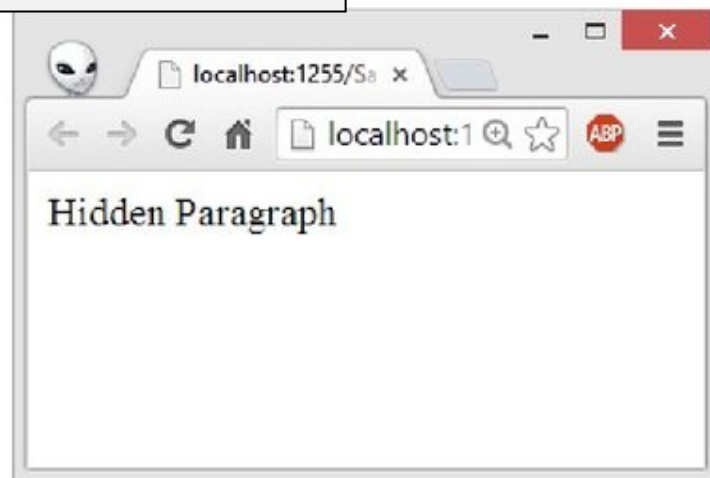
```
document.getElementById(id).attribute = Nova vrijednost;
```

```
<html>
<body>
  <p id="target" hidden>Hidden Paragraph</p>
  <script>
    document.getElementById('target').hidden = '';
  </script>
</body>
</html>
```



```
document.getElementById(id).style.property = Nova vrijednost;
```

```
<html>
<body>
  <p id="target" style="display: none">Hidden
Paragraph</p>
  <script>
    document.getElementById('target').style.display = '';
  </script>
</body>
</html>
```



# HTML5 nove funkcionalnosti

- Proširenja, potpuno novi alati i funkcionalnosti u HTML5:
  - Multimedia API
  - Offline Web Applications
  - Drag and Drop API
  - History API
  - HTML5 Microdata
  - HTML5 Storage API
  - Geolocation API
  - Web workers API
  - WebSocket API
  - Canvas

# Object Canvas (1)

- Platno predstavlja područje unutar kojeg je omogućeno crtanje grafičkih oblika korištenjem JavaScript koda te ima fiksnu veličinu – dužinu i širinu. JavaScript služi kao medij kojim se crta i animira kompleksna grafika.
- Za upotrebu platna nisu potrebne dodatne biblioteke već kompatibilan web preglednik i uređivač kôda koji se koristi.
- Platno koristi WebGL (Web Graphics Library) za 3D grafiku web stranica unutar bilo kojeg kompatibilnog web preglednika bez korištenja dodatnih priključaka (*plug-in*).

## Object Canvas (2)

- Objekt Canvas se može koristiti za vizualizaciju podataka, web aplikacije, animiranu grafiku ili igrice.
- Canvas je pravokutnik na web stranici s definiranom visinom i dužinom po kojemu je pomoću JavaScripta moguće crtati. Prema zadanim postavkama širina Canvasa je 300 piksela, a visina 150 piksela.
- JavaScript može pristupiti objektu Canvas i manipulirati ga kroz velik broj podržanih funkcija.
  - Omogućuju crtanje što dalje omogućuje dinamičko generiranje grafike.

# Object Canvas (3)

- Temelji se na sustavu bitmapa
  - Sve što je iscrtano predstavlja jednu jedinstvenu sliku
  - Svaka promjena zahtijeva ponovno crtanje slike koja se prikazuje.
- Prije Canvas elementa web preglednici su koristili SVG za crtanje unutar web stranica
- Za razliku od Canvasa, SVG koristi vektorski sustav koji elemente crta kao odvojene DOM objekte, a kod promjene SVG objekta web preglednik će automatski promijeniti objekt.
- Kod Canvas elementa dostupna su samo dva atributa i tri metode:

## Object Canvas (4)

- Canvas koristi standardni koordinatni sustav gdje se koordinata (0,0) nalazi u gornjem lijevom kutu.
- Duž x osi vrijednost koordinata se povećava prema desnom rubu canvasa, dok vrijednost koordinata y osi raste prema donjem rubu canvasa.
- Svaki Canvas pojedinač (element) ima kontekst crtanja.
- Kontekst crtanja je mjesto na kojemu su definirane sve metode i svojstva crtanja. Kontekst crtanja pruža API za crtanje raznih oblika i teksta, prikazivanje slika, manipulaciju slika, ...

```
var moj_canvas = document.getElementById("mojCanvas");  
var moj_kontekst = moj_canvas.getContext("2d");
```

## Object Canvas (5)

- Dimenzije se zadaju u konstrukuru Canvas objekta ili korištenjem *width* i *height* atributa

```
moj_canvas.width = 480;  
moj_canvas.height = 200;
```

- Potrebno je obrisati područje Canvas elementa za ponovno crtanje
  - `clearRect` (x koordinata, y koordinata, širina, visina)
  - Promjenom dimenzija Canvas objekta sva svojstva konteksta postavljaju se na početne pretpostavljane vrijednosti

# Crtanje pravokutnika (1)

- Crtanje pravokutnika postiže se jednom od metoda kao što je `fillRect` (x koordinata, y koordinata, širina, visina) koja crta ispunjen pravokutnik.
  - Ako stil ispune nije drugačije zadan, pravokutnik uzima zadani stil, tj. crnu ispunu.
- Ostale često korištene metode:
  - **`rect()`** – stvara pravokutnik, za crtanje koriste se metode **`stroke()`** i **`fill()`**
  - **`strokeRect()`** – crta pravokutnik bez ispune
  - **`clearRect()`** – briše sadržaj pravokutnika

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.beginPath();  
ctx.rect(20, 20, 150, 100);  
ctx.stroke();
```



# Crtanje pravokutnika (2)

```
<!DOCTYPE html>
<html>
<body>
  <canvas id="myCanvas" width="300" height="150" style="border:1px
solid #d3d3d3;">
    Your browser does not support the HTML5 canvas tag.</canvas>
  <script>
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");
    ctx.fillRect(20, 20, 150, 100);
  </script>
</body>
</html>
```



# Stilovi, boje i sjene (1)

- Svaki kontekst crtanja pamti svojstva sve dok je web stranica otvorena ili se ponovno ne pokrene.
- Vrijednost svojstva konteksta crtanja zadaje se prije crtanja na Canvas.
  - Ako se ne zada vrijednost, koristi se zadana vrijednost ili posljednja zadana vrijednost.
  - Svojstvo `fillStyle` može biti boja, uzorak ili gradijent koji se koristi za ispunu, a početna zadana vrijednost za `fillStyle` je crna boja.

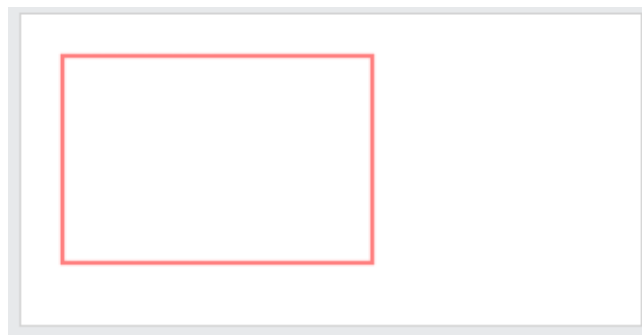
```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.fillStyle = "#FF0000";  
ctx.fillRect(20, 20, 150, 100);
```



## Stilovi, boje i sjene (2)

- Svojstvo `strokeStyle` može biti boja, uzorak ili gradijent koji se koristi za crtanje rubova, a početna zadana vrijednost za `strokeStyle` je crna boja.

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.strokeStyle = "#FF0000";  
ctx.strokeRect(20, 20, 150, 100);
```



- Svojstvo `shadowColor` određuje boju koja će se koristiti kao sjena, a početna zadana vrijednost za `shadowColor` je crna boja.

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.shadowBlur = 20;  
ctx.shadowColor = "black";  
ctx.fillStyle = "red";  
ctx.fillRect(20, 20, 100, 80);
```



## Stilovi, boje i sjene (3)

- Metoda `createLinearGradient` (x koordinata, y koordinata, x koordinata, y koordinata) stvara linearni gradijent koji se može iskoristiti za ispunu pravokutnika, krugova, linija, teksta i ostalih oblika. Vrijednosti koje se proslijeđuju metodi su početna koordinata gradijenta i završna koordinata gradijenta.
- Metodom `addColorStop` (pozicija, boja) određuje boju i položaj gradijenta.

```
var c = document.getElementById('myCanvas');
```

```
var ctx = c.getContext('2d');
```

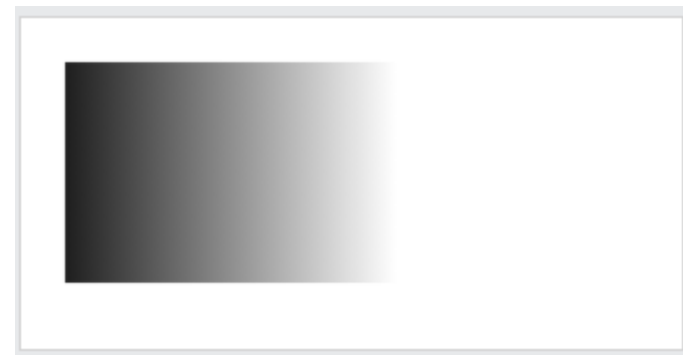
```
var grd = ctx.createLinearGradient(0, 0, 170, 0);
```

```
grd.addColorStop(0, "black");
```

```
grd.addColorStop(1, "white");
```

```
ctx.fillStyle = grd;
```

```
ctx.fillRect(20, 20, 150, 100);
```



## Stilovi, boje i sjene (4)

- Ostale metode za stvaranje stilova, boja i sjena su `createPattern()` i `createRadialGradient()`
  - Metoda `createPattern()` ponavlja navedeni element u određenom smjeru. Taj element može biti slika, video ili neki drugi Canvas. Ponavljajući element može se koristiti kao ispunja pravokutnika, krugova, linija, teksta i ostalih oblika.
  - Metoda `createRadialGradient()` stvara kružni gradijent, a metoda `addColorStop()` određuje boje i njihove pozicije na gradijentu.

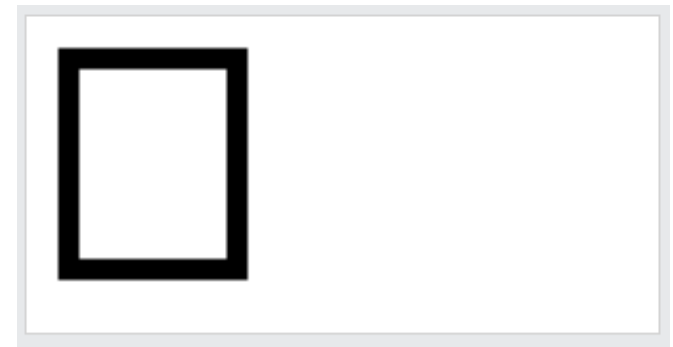
```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
var img = document.getElementById("lamp");  
var pat = ctx.createPattern(img, "repeat");  
ctx.rect(0, 0, 150, 100);  
ctx.fillStyle = pat;  
ctx.fill();
```



# Stilovi linija

- Svojstvo `lineWidth` zadaje trenutnu širinu linije u pikselima, a zadana vrijednost za `lineWidth` je 1.
- Ostala svojstva stilova linija su `lineCap`, `lineJoin` i `miterLimit`. Svojstvo `lineCap` zadaje stil završetka linija, svojstvo `lineJoin` zadaje tip ugla koji se stvara kada se dvije linije susretnu, a svojstvo `miterLimit` zadaje maksimalnu dužinu između unutarnjeg ugla i vanjskog ugla kada se dvije linije susretnu.

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.lineWidth = 10;  
ctx.strokeRect(20, 20, 80, 100);
```



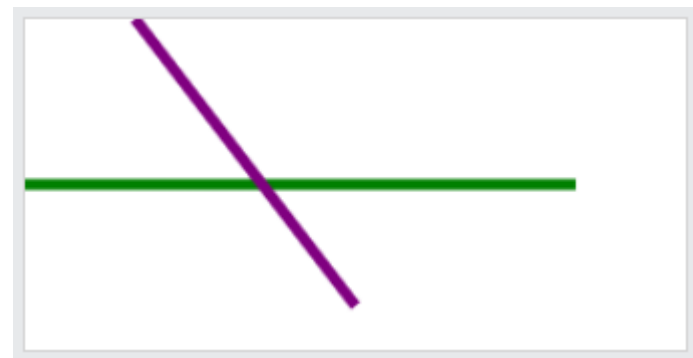
# Putanje

- Metoda `beginPath()` započinje putanju ili resetira trenutnu putanju linije.
- Metoda `moveTo (x koordinata, y koordinata)` određuje početnu koordinatu linije, `lineTo (x koordinata, y koordinata)` zadaje krajnju koordinatu linije.
- Metoda `stroke()` na kraju crta zadanu liniju.

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");
```

```
ctx.beginPath();  
ctx.lineWidth = "5";  
ctx.strokeStyle = "green"; // Green path  
ctx.moveTo(0, 75);  
ctx.lineTo(250, 75);  
ctx.stroke(); // Draw it
```

```
ctx.beginPath();  
ctx.strokeStyle = "purple"; // Purple path  
ctx.moveTo(50, 0);  
ctx.lineTo(150, 130);  
ctx.stroke(); // Draw it
```



# Tekst (1)

- Za ispis teksta u objekt Canvas prvo je potrebno odrediti font koji će se za to koristiti i postaviti njegova svojstva (barem veličina).
- Nakon što je font definiran koristi se `fillText()` metoda za ispis teksta.
- Umjesto korištenja `fillText()` metode mogu se koristiti i druge metode za ispis i formatiranje teksta:
  - `strokeText()` – crta konture oko teksta
  - `strokeStyle` – definira boju konture
  - `lineWidth` – definira debljinu konture
  - `measureText()` – vraća objekt sa širinom zadanog teksta
- Svojstva za crtanje teksta su `textAlign` i `textBaseline`.
  - Svojstvo `textAlign` zadaje poravnanje za tekstualni sadržaj prema zadanoj točki sidrišta.
  - Svojstvo `textBaseline` zadaje tekstualnu osnovicu koja se rabi pri izradi teksta.



## Tekst (2)

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");
```

```
ctx.font = "20px Georgia";  
ctx.fillText("Hello World!", 10, 50);
```

```
ctx.font = "30px Verdana";  
// Create gradient  
var gradient = ctx.createLinearGradient(0, 0, c.width, 0);  
gradient.addColorStop("0", "magenta");  
gradient.addColorStop("0.5", "blue");  
gradient.addColorStop("1.0", "red");  
// Fill with gradient  
ctx.fillStyle = gradient;  
ctx.fillText("Big smile!", 10, 90);
```

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.font = "30px Arial";  
var txt = "Hello World"  
ctx.fillText("width:" + ctx.measureText(txt).width, 10, 50)  
ctx.fillText(txt, 10, 100);
```

Hello World!

Big smile!

width:154.4970703125

Hello World

# Tekst (3)

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");
```

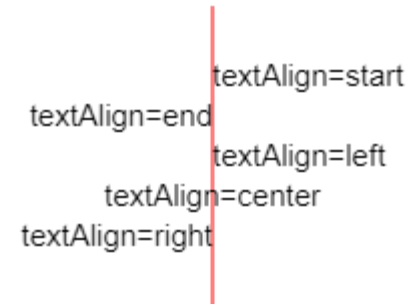
```
// Create a red line in position 150
```

```
ctx.strokeStyle = "red";  
ctx.moveTo(150, 20);  
ctx.lineTo(150, 170);  
ctx.stroke();
```

```
ctx.font = "15px Arial";
```

```
// Show the different textAlign values
```

```
ctx.textAlign = "start";  
ctx.fillText("textAlign=start", 150, 60);  
ctx.textAlign = "end";  
ctx.fillText("textAlign=end", 150, 80);  
ctx.textAlign = "left";  
ctx.fillText("textAlign=left", 150, 100);  
ctx.textAlign = "center";  
ctx.fillText("textAlign=center", 150, 120);  
ctx.textAlign = "right";  
ctx.fillText("textAlign=right", 150, 140);
```



# Tekst (4)

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");
```

```
//Draw a red line at y = 100
```

```
ctx.strokeStyle = "red";
```

```
ctx.moveTo(5, 100);
```

```
ctx.lineTo(395, 100);
```

```
ctx.stroke();
```

```
ctx.font = "20px Arial"
```

```
//Place each word at y = 100 with different textBaseline values
```

```
ctx.textBaseline = "top";
```

```
ctx.fillText("Top", 5, 100);
```

```
ctx.textBaseline = "bottom";
```

```
ctx.fillText("Bottom", 50, 100);
```

```
ctx.textBaseline = "middle";
```

```
ctx.fillText("Middle", 120, 100);
```

```
ctx.textBaseline = "alphabetic";
```

```
ctx.fillText("Alphabetic", 190, 100);
```

```
ctx.textBaseline = "hanging";
```

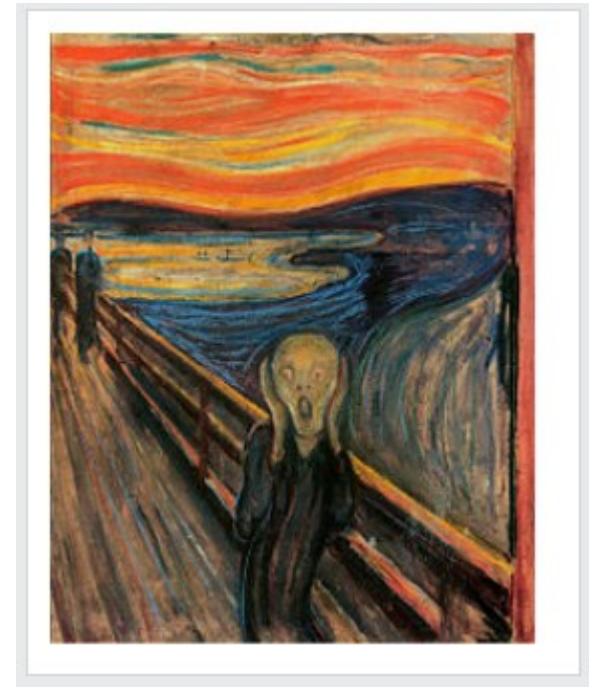
```
ctx.fillText("Hanging", 290, 100);
```

A horizontal red line is shown. Above the line, the words "Bottom", "Middle", and "Alphabetic" are written. Below the line, the words "Top" and "Hanging" are written. This illustrates the different text baselines used in the code.

## Crtanje slika (2)

- Metoda `drawImage` (objekt slike, x koordinata, y koordinata) omogućuje crtanje slika, videa ili drugih Canvas elemenata.
  - Metodi se proslijeđuje objekt koji se crta i koordinate gornjeg lijevog kuta objekta.

```
window.onload = function() {  
    var c = document.getElementById("myCanvas");  
    var ctx = c.getContext("2d");  
    var img = document.getElementById("scream");  
    ctx.drawImage(img, 10, 10);  
};
```



## Crtanje slika (3)

- Prije pozivanja `drawImage()` metode potrebno je definirati sliku.
- HTML5 pruža tri načina dobivanja slike za crtanje.
  1. Pomoću metode `createImageData()`
    - Sporo jer se objekt kreira prije korištenja.
  2. Korištenjem `<img>` HTML elementa navedenog u stranici te kopiranje te slike u Canvas objekt

```

```

```
var img = document.getElementById("imageCrop");
context.drawImage(img, 10, 10);
```
  3. Učitavanjem slike iz datoteke na disku
    - Potrebno je pričekati da se slika učitava u potpunosti (`onload`)

## Crtanje slika (4)

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var imgData = ctx.createImageData(100, 100);
var i;
for (i = 0; i < imgData.data.length; i += 4) {
    imgData.data[i+0] = 255;
    imgData.data[i+1] = 0;
    imgData.data[i+2] = 0;
    imgData.data[i+3] = 255;
}
ctx.putImageData(imgData, 10, 10);
```



### Određivanje boje (R, G, B, Alpha)

R = 0...255, crvena komponenta

G = 0...255, zelena komponenta

B = 0...255, plava komponenta

Alpha = 0...255, prozirnost (255 potpuno vidljiv)

# Geolocation API (1)



- Dohvaća zemljopisni položaj preglednika
  - Ako ga je moguće odrediti
  - Ako postoji dozvola za pristup podacima
- Najvažnija sučelja:
  - **Geolocation** – osnovni razred
  - **GeolocationPosition** – predstavlja položaj korisnika
  - **GeolocationCoordinates** – koordinate korisnika
  - **GeolocationPositionError** – greška položaja
  - **Navigator.geolocation** – ulazna točka u Geolocation API, vraća instancu Geolocation razreda

# Geolocation API (2)



```
navigator.geolocation.getCurrentPosition(  
  function(position) {  
    position.coords.latitude;  
    position.coords.longitude;  
  }, function(error) {  
    // Šifra greške:  
    // 0: unknown error  
    // 1: permission denied  
    // 2: position unavailable  
    // 3: timed out  
  });
```

```
navigator.geolocation.watchPosition(function(position) {  
  // Prati položaj ako se promijeni  
});
```



# Geolocation API (3)

- Drugi argument u metodi `getCurrentPosition()` koristi se za obradu grešaka:

```
function showError(error) {  
    switch(error.code) {  
        case error.PERMISSION_DENIED:  
            x.innerHTML = "User denied the request for Geolocation."  
            break;  
        case error.POSITION_UNAVAILABLE:  
            x.innerHTML = "Location information is unavailable."  
            break;  
        case error.TIMEOUT:  
            x.innerHTML = "The request to get user location timed out."  
            break;  
        case error.UNKNOWN_ERROR:  
            x.innerHTML = "An unknown error occurred."  
            break;  
    }  
}
```

# Geolocation API (4)

- U slučaju uspjeha metoda `getCurrentPosition()` vraća objekt koji ima sljedeća svojstva:
  - `coords.latitude` – zemljopisna širina
  - `coords.longitude` – zemljopisna dužina
  - `coords.accuracy` – točnost
  - `coords.altitude` – visina (m), ako je podatak dostupan
  - `coords.altitudeAccuracy` – točnost podatka o visini (m)
  - `coords.heading` – smjer kretanja (°)
  - `coords.speed` – brzina (m/s)
  - `Timestamp` – datum i vrijeme

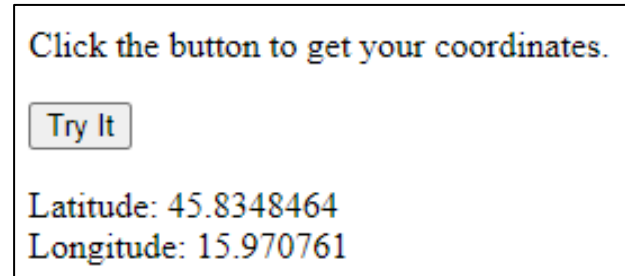
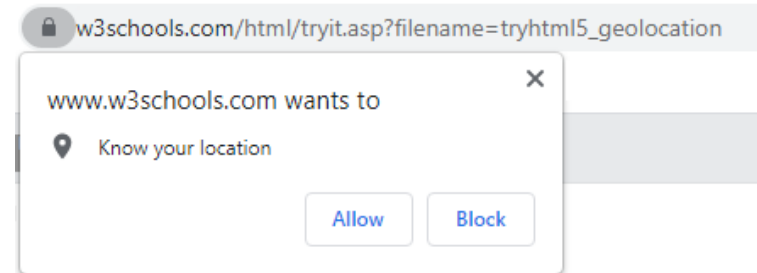
# Geolocation API – osnovni primjer

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>

<script>
var x = document.getElementById("demo");

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>
</body>
</html>
```



# Geolocation API – Google Maps (1)



```
let map, infoWindow;
```

Početna lokacija prikaza karte i razina zooma

```
function initMap() {  
  map = new google.maps.Map(document.getElementById("map"), {  
    center: { lat: -34.397, lng: 150.644 },  
    zoom: 10,  
  });  
  infoWindow = new google.maps.InfoWindow();  
  
  const locationButton = document.createElement("button");  
  
  locationButton.textContent = "Pan to Current Location";  
  locationButton.classList.add("custom-map-control-button");  
  map.controls[google.maps.ControlPosition.TOP_CENTER].push(locationButton);  
  locationButton.addEventListener("click", () => {
```

```
    // Try HTML5 geolocation.
```

```
    if (navigator.geolocation) {  
      navigator.geolocation.getCurrentPosition(  
        (position) => {  
          const pos = {  
            lat: position.coords.latitude,  
            lng: position.coords.longitude,  
          };  
  
          infoWindow.setPosition(pos);  
          infoWindow.setContent("Location found.");  
          infoWindow.open(map);  
          map.setCenter(pos);  
        },  
        () => {  
          handleLocationError(true, infoWindow, map.getCenter());  
        }  
      );  
    } else {  
      // Browser doesn't support Geolocation  
      handleLocationError(false, infoWindow, map.getCenter());  
    }  
  });  
}
```

Dohvaćanje trenutnog položaja

Obrada greške

Pomicanje karte

```
function handleLocationError(browserHasGeolocation,  
  infoWindow, pos) {  
  infoWindow.setPosition(pos);  
  infoWindow.setContent(  
    browserHasGeolocation  
      ? "Error: The Geolocation service failed."  
      : "Error: Your browser doesn't support geolocation."  
  );  
  infoWindow.open(map);  
}
```

# Geolocation API – Google Maps (2)



- Primjer1\_TrenutnaLokacija
  - Primjer2\_Lokalizacija
  - Primjer3\_DogađajiMarkeri1
  - Primjer4\_DogađajiMarkeri2
  - Primjer5\_DogađajiMarkeri3
- 
- Izvorni kod nalazi se na stranicama predmeta

# Geolocation API – OpenStreetMaps



```
function geoFindMe() {

    const status = document.querySelector('#status');
    const mapLink = document.querySelector('#map-link');

    mapLink.href = '';
    mapLink.textContent = '';

    function success(position) {
        const latitude = position.coords.latitude;
        const longitude = position.coords.longitude;

        status.textContent = '';
        mapLink.href = `https://www.openstreetmap.org/#map=18/${latitude}/${longitude}`;
        mapLink.textContent = `Latitude: ${latitude} °, Longitude: ${longitude} °`;
    }

    function error() {
        status.textContent = 'Unable to retrieve your location';
    }

    if(!navigator.geolocation) {
        status.textContent = 'Geolocation is not supported by your browser';
    } else {
        status.textContent = 'Locating...';
        navigator.geolocation.getCurrentPosition(success, error);
    }
}

document.querySelector('#find-me').addEventListener('click', geoFindMe);
```

```
<button id = "find-me">Show my location</button><br/>
<p id = "status"></p>
<a id = "map-link" target="_blank"></a>
```

# Geolocation API – OpenStreetMaps



```
function geoFindMe() {
```

```
    const status = document.querySelector('#status');  
    const mapLink = document.querySelector('#map-link');
```

```
    mapLink.href = '';  
    mapLink.textContent = '';
```

```
    function success(position) {  
        const latitude = position.coords.latitude;  
        const longitude = position.coords.longitude;
```

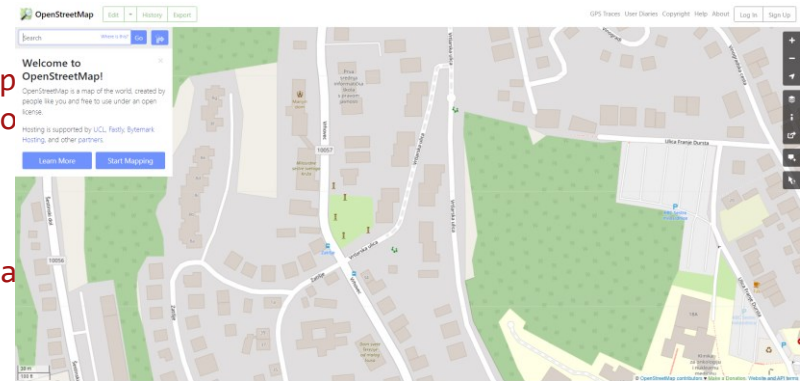
```
        status.textContent = '';  
        mapLink.href = `https://www.openstreetmap.org/#map`  
        mapLink.textContent = `Latitude: ${latitude} °, Longitude: ${longitude} °`  
    }
```

```
    function error() {  
        status.textContent = 'Unable to retrieve your location';  
    }
```

```
    if(!navigator.geolocation) {  
        status.textContent = 'Geolocation is not supported by your browser';  
    } else {  
        status.textContent = 'Locating...';  
        navigator.geolocation.getCurrentPosition(success, error);  
    }
```

```
    document.querySelector('#find-me').addEventListener('click', geoFindMe);
```

```
<button id = "find-me">Show my location</button><br/>  
<p id = "status"></p>  
<a id = "map-link" target="_blank"></a>
```



Show my location

Latitude: 45.8181636 °, Longitude: 15.9486195 °

# Web Storage API (1)



- HTML5 stranice mogu pohranjivati podatke lokalno u web pregledniku
  - Podaci se razmjenjuju samo na zahtjev, a ne kod svakog HTML *requesta*
  - Podaci su strukturirani kao uređeni parovi naziv/vrijednost (*name/value pair*)
  - Web stranica može pristupiti podacima koje je prethodno pohranila, a ne i od drugih web stranica
  - *Storage limit* je puno veći (>5MB)
  - Podaci se nikad ne prenose na poslužitelj
- Sigurniji i brži mehanizam od kolačića (*cookies*)



# Web Storage API (2)



- Nova svojstva objekta **Window** za pohranu podataka:
  - **window.localStorage**
    - Trajna pohrana, bez vremena isteka
  - **window.sessionStorage**
    - Podaci perzistiraju samo unutar sesije, podaci se brišu kada se prozor (ili tab) web preglednika zatvori
- Događaj:
  - **StorageEvent**
    - Generira se nakon svake promjene prostora za pohranu (*storage area*)
  - **WindowEventHandlers.onstorage**
    - Metoda za obradu događaja, npr. pohrana novog zapisa

# Web Storage API (3)



- Prije korištenja dobro je provjeriti da li preglednik podržava HTML5 Web Storage API:

```
if (typeof (Storage) !== "undefined") {  
    // Programski kod za localStorage i sessionStorage  
}  
else {  
    // HTML5 Web Storage funkcionalnost nije podržana  
}
```

- 'Incognito', 'Private Browsing'
  - Svi podaci se brišu nakon zatvaranja **preglednika**, ne prozora ili taba
  - Različite implementacije ovisno o pregledniku

# Web Storage API – primjer (1)

## ■ Primjer uporabe:

### Sučelje Storage

- .setItem() // Pohrani (setter)
- .getItem() // Dohvati/pročitaj (getter)
- .removeItem() // Izbriši zapis
- .clear() // Izbriši sve zapise

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function countClicks() {
        if (localStorage.clickcount) {
          localStorage.clickcount = Number(localStorage.clickcount) + 1;
        } else {
          localStorage.clickcount = 1;
        }
        document.getElementById('target').innerHTML = localStorage.clickcount;
      }
    </script>
  </head>
  <body>
    <p>You have clicked the button <span id='target'></span> time(s).</p>
    <button type="button" onclick=countClicks()>
      Count
    </button>
  </body>
</html>
```

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {  
    populateStorage();  
} else {  
    setStyles();  
}  
  
function populateStorage() {  
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);  
    localStorage.setItem('font', document.getElementById('font').value);  
    localStorage.setItem('image', document.getElementById('image').value);  
  
    setStyles();  
}
```

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {  
    populateStorage();  
} else {  
    setStyles();  
}  
  
function populateStorage() {  
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);  
    localStorage.setItem('font', document.getElementById('font').value);  
    localStorage.setItem('image', document.getElementById('image').value);  
  
    setStyles();  
}
```

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {
    populateStorage();
} else {
    setStyles();
}

function populateStorage() {
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);
    localStorage.setItem('font', document.getElementById('font').value);
    localStorage.setItem('image', document.getElementById('image').value);
}

function setStyles() {
    var currentColor = localStorage.getItem('bgcolor');
    var currentFont = localStorage.getItem('font');
    var currentImage = localStorage.getItem('image');

    document.getElementById('bgcolor').value = currentColor;
    document.getElementById('font').value = currentFont;
    document.getElementById('image').value = currentImage;

    htmlElem.style.backgroundColor = '#' + currentColor;
    pElem.style.fontFamily = currentFont;
    imgElem.setAttribute('src', currentImage);
}
```

# Web SQL Database API (1)



- Web SQL Database API
  - Skup programskih poziva za korištenje baze podataka na klijentu pomoću SQL operacija
  - Zasebna specifikacija, nije dio HTML5 specifikacije
- Najvažnije metode:
  - **openDatabase** – koristi postojeću bazu podataka ili stvara novi objekt baze podataka
  - **transaction** – upravljanje transakcijama, *commit*, *roll back*
  - **executeSql** – izvršava SQL upit
- Otvaranje baze podataka:

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

Naziv baze podataka

Verzija

Opis

Veličina u bajtovima

Objekt instance baze podataka (*callback*)

# Web SQL Database API (2)



- Kreiranje tablice (CREATE TABLE):
  - Nakon otvaranje baze db, ako ne postoji potrebno je kreirati barem jednu tablicu

```
db.transaction(function (tx) {  
    tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');  
});
```

- Unos podataka (INSERT):

```
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "predavanje")');
```

- Upit (SELECT):

```
tx.executeSql('INSERT INTO LOGS (id,log) VALUES (?, ?)',  
[e_id, e_log];
```

Vanjske varijable



# Web SQL Database API – primjer

```
<!DOCTYPE HTML>
```

```
<html>
```

```
  <head>
```

```
    <script type = "text/javascript">
```

```
      var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

```
      var msg;
```

```
      db.transaction(function (tx) {
```

```
        tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
```

```
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
```

```
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
```

```
        msg = '<p>Log message created and row inserted.</p>';
```

```
        document.querySelector('#status').innerHTML = msg;
```

```
      })
```

```
      db.transaction(function (tx) {
```

```
        tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
```

```
          var len = results.rows.length, i;
```

```
          msg = "<p>Found rows: " + len + "</p>";
```

```
          document.querySelector('#status').innerHTML += msg;
```

```
          for (i = 0; i < len; i++) {
```

```
            msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
```

```
            document.querySelector('#status').innerHTML += msg;
```

```
          }
```

```
        }, null);
```

```
      });
```

```
    </script>
```

```
  </head>
```

```
  <body>
```

```
    <div id = "status" name = "status">Status Message</div>
```

```
  </body>
```

```
</html>
```

# Web SQL Database API – primjer

```
<!DOCTYPE HTML>
```

```
<html>
```

```
  <head>
```

```
    <script type = "text/javascript">
```

```
      var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

```
      var msg;
```

```
      db.transaction(function (tx) {
```

```
        tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
```

```
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
```

```
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
```

```
        msg = '<p>Log message created and row inserted.</p>';
```

```
        document.querySelector('#status').innerHTML = msg;
```

```
      })
```

```
      db.transaction(function (tx) {
```

```
        tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
```

```
          var len = results.rows.length, i;
```

```
          msg = "<p>Found rows: " + len + "</p>";
```

```
          document.querySelector('#status').innerHTML += msg;
```

```
          for (i = 0; i < len; i++) {
```

```
            msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
```

```
            document.querySelector('#status').innerHTML += msg;
```

```
          }
```

```
        }, null);
```

```
      });
```

```
    </script>
```

```
  </head>
```

```
  <body>
```

```
    <div id = "status" name = "status">Status Messa
```

```
  </body>
```

```
</html>
```

Log message created and row inserted.

Found rows: 2

**foobar**

**logmsg**

# Mikropodaci (1)

- Mikropodaci (*microdata*) predstavljaju strukture podataka koje se koriste unutar HTML5 web stranica
  - Definirane od strane korisnika – zapisuju različite podatke i tipove podataka, imaju vlastiti rječnik i strukturu
  - Generička primjena, može ih biti proizvoljno mnogo, nisu unificirane, podaci su pomiješani sa HTML5 kodom
  - Jednostavan način za opis neke semantike na strukturiran način

```
<html>
<body>
  <div itemscope>
    <p>My name is <span itemprop = "name">Ivan</span>.</p>
  </div>
  <div itemscope>
    <p>My name is <span itemprop = "name">Ivić</span>.</p>
  </div>
</body>
</html>
```

Primjer

Ključna riječ **itemscope**

Ključna riječ **itemprop**

Uređeni par (**itemscope** , **itemprop**)

## Mikropodaci (2)

- Korištenjem mikropodataka definiraju se grupe uređenih parova zapisa (*items*) (naziv, svojstvo)
- Globalni atributi koji se mogu koristiti s bilo kojim HTML5 elementom:
  - **itemid** – globalni i jedinstveni identifikator zapisa
  - **itemscope** – naziv, stvara novi zapis (*item*)
  - **itemprop** – svojstvo, dodaje svojstvo (*property*) zapisu
  - **itemref** – svojstva koje nisu nasljednici elementa definiranih atributom **itemscope** mogu se pridružiti zapisu pomoću **itemref** atributa
  - **itemtype** – određuje URL rječnika kojim su definirana svojstva zapisa definiranih atributom **itemprop**.

# Mikropodaci – primjer (1)

```
<div itemscope itemtype="http://schema.org/Images">
  <img itemprop = "imageUniZg" src = " https://www.fer.unizg.hr/_pub/
themes_static/fer2016/default/img/UniZg_logo.png" alt = "UniZG_logo">
  <img itemprop = "imageFER" src = " https://www.fer.unizg.hr/_pub/th
emes_static/fer2016/default/img/FER_logo.png" alt = "FER_logo">
</div>
```

```
<html>
  <body>
    <div itemscope>
      Početak semestra:
      <time itemprop = "semester_start" datetime = "2021-10-04">
        4. listopada 2021.
      </time>
    </div>
  </body>
</html>
```

**Result**

Početak semestra: 4. listopada 2021.

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -

      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

      <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
        RATING:
        <span itemprop="ratingValue">4.6</span> (
        <span itemprop="ratingCount">8864</span> ratings )
      </div>

      <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
        Price: $<span itemprop="price">1.00</span>
        <meta itemprop="priceCurrency" content="USD" />
      </div>
    </div>
  </body>
</html>
```

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -

      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

      <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
        RATING:
        <span itemprop="ratingValue">4.6</span> (
        <span itemprop="ratingCount">8864</span> ratings )
      </div>

      <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
        Price: $<span itemprop="price">1.00</span>
        <meta itemprop="priceCurrency" content="USD" />
      </div>
    </div>
  </body>
</html>
```

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -

      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="applicationCategory" href="http://schema.org/GameApplication" />
      <div itemprop="aggregateRating" schema="http://schema.org/AggregateRating">
        RATING:
        <span itemprop="ratingValue">4.6</span>
        <span itemprop="ratingCount">8864</span> ratings )
      </div>

      <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
        Price: $<span itemprop="price">1.00</span>
        <meta itemprop="priceCurrency" content="USD" />
      </div>
    </div>
  </body>
</html>
```

**Structured Data Testing Tool**  
Googleov *online* alat za izdvajanje strukture mikropodataka iz HTML koda web stranice  
<https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data>

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00



# Mikropodaci – širi kontekst

- Semantičke web tehnologije
  - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
  - *Napredni modeli i baze podataka, FER3*
- Resource Description Framework (RDF)
  - Služi za zapis „podataka o podacima” ili „znanja”
  - temelji se na konceptu trojki: subjekt – predikat – objekt
    - subjekt ima određeno svojstvo (predikat) čija je vrijednost objekt
    - subjekt je uvijek resurs, dok objekt može biti resurs ili podatkovna
    - vrijednost
  - RDF trojka predstavlja izjavu
    - serijalizirana u XML, trojna notacija (N3) ili Turtle format
    - najčešće prikazana u obliku graf
- Resource Description Framework in Attributes (RDFa)
  - W3C standard za proširenje XHTML-a s umetanjem izjava u RDF-u
  - Pojednostavljeno „izvlačenje” podataka iz HTML stranica

# Mikropodaci – širi kontekst

- Semantičke web tehnologije
  - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
  - *Napredni modeli i baze podataka, FER3*
- Resource Description Framework (RDF)
  - Služi za zapis „podataka o podacima” ili „znanja”
  - temelji se na konceptu trojki: subjekt – predikat – objekt
    - subjekt ima određeno svojstvo (predikat) čija je vrijednost objekt
    - subjekt je uvijek resurs, dok objekt može biti resurs ili podatkovna
    - vrijednost
  - RDF trojka predstavlja izjavu
    - serijalizirana u XML, trojna notacija (N3) ili Turtle format
    - najčešće prikazana u obliku graf
- Resource Description Framework in Attributes (RDFa)
  - W3C standard za proširenje XHTML-a s umetanjem izjava u RDF-u
  - Pojednostavljeno „izvlačenje” podataka iz HTML stranica

# Mikropodaci – širi kontekst

- Semantičke web tehnologije
  - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
  - *Napredni modeli i baze podataka, FER3*
- Resource Description Framework (RDF)
 

HTML

  - Služi za zapis „podataka“
  - temelji se na konceptu trojke (subjekt – predikat – objekt)
    - subjekt ima određenu vrijednost
    - subjekt je uvijek resurs ili podatkovna vrijednost
    - vrijednost je vrijednost objekt
  - RDF trojka predstavlja izjavu
    - serijalizirana u XML, trojna notacija (N3) ili Turtle format
    - najčešće prikazana u obliku HTML5 + RDFa

```
<p>
  Google Inc.<br>
  P.O. Box 1234<br>
  Mountain View, CA<br>
  94043<br>
  United States<br>
</p>
```
- Resource Description Framework (RDFa)
 

HTML5 + RDFa

  - W3C
  - Poj

```
<p vocab="http://Schema.org/" typeof="PostalAddress"><br>
  <span property="name">Google Inc.</span><br>
  P.O. Box <span property="postOfficeBoxNumber">1234</span><br>
  <span property="addressLocality">Mountain View</span>,<br>
  <span property="addressRegion">CA</span><br>
  <span property="postalCode">94043</span><br>
  <span property="addressCountry">United States</span><br>
</p>
```

# Web Workers API (1)

- Tipično prilikom izvršavanja različitih skripti unutar HTML stranice web preglednik prestaje reagirati na naredbe korisnika („zamrzne se”) sve dok skripta ne završi.
- Web radnici (*Web workers*) su jednostavan način za izvršavanja JavaScript programskog koda u zasebnoj dretvi neovisno od prikaza web stranice.
  - Web worker dretva može se izvršavati bez ometanja rada korisničkog sučelja web preglednika.
  - Obično se koriste za implementaciju složenih i procesorski zahtjevnih zadataka.
  - Izvršavaju se asinkrono na klijentu.

## Web Workers API (2)

- Prvo je potrebno provjeriti da li web preglednik podržava Web Workers `if (typeof(Worker) !== "undefined")`
- Izvorni kod Web Workera nalazi se u zasebnoj JavaScript datoteci (npr. `demo_workers.js`)
  - Web Worker šalje podatke u HTML stranicu metodom `postMessage(var);`
- ...iz koje se kreira Web Worker objekt u HTML stranici

```
if (typeof(w) == "undefined") {  
    w = new Worker("demo_workers.js");  
}  
w.onmessage = function(event){  
    document.getElementById("result").innerHTML = event.data;  
};
```

- Nakon toga HTML stranica može primiti i slati podatke prema Web Workeru
- Objekt Web Worker se može uništiti (i osloboditi zauzeta memorija) `w.terminate();`
- ...ili ponovno koristiti kasnije u kôdu `w = undefined;`

# Web Workers API – primjer



WEB WORKERS

```
<!DOCTYPE html>
<html>
<body>

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
var w;

function startWorker() {
  if (typeof(Worker) !== "undefined") {
    if (typeof(w) == "undefined") {
      w = new Worker("demo_workers.js");
    }
    w.onmessage = function(event) {
      document.getElementById("result").innerHTML = event.data;
    };
  } else {
    document.getElementById("result").innerHTML = "Sorry! No Web Worker support.";
  }
}

function stopWorker() {
  w.terminate();
  w = undefined;
}
</script>

</body>
</html>
```

# WebSocket API (1)

- WebSocket API omogućuje otvaranje dvosmjerne (*full-duplex*) komunikacijske sesije između web preglednika na klijentu i poslužitelja
  - WebSocket je transportni protokol (OSI 7) koji se odvija putem TCP *socketa*
  - Slanje poruka poslužitelju i primanje odgovore obavlja se temeljem događaja (*event-based*) bez potrebe slanja klijentskih zahtjeva i odgovora sa poslužitelja (*polling*)
  - Koristi jednu klijentsku utičnicu (*socket*) dostupnu kroz standardizirano JavaScript sučelje u preglednicima koji podržavaju HTML5
- Koristi se pri izradi web aplikacija za rad u bliskom stvarnom vremenu:
  - Cijene dionica, financijska tržišta, ...
  - *Chat*
  - Interakcija s korisnicima u stvarnom vremenu
  - Praćenje položaja (*live location tracking*)
  - IoT
  - Upravljanje WebRTC pozivima

# WebSocket API (2)

Klijent, ekstenzija Chrome web preglednika



Konstruktor:

```
var Socket = new WebSocket(url, [protocol] );
```

**url** = URL na koji se utičnica povezuje

**protocol** = opcionalni argument, pod-protokol poslužitelja

WebSocket Tester

Supports both,  
ws and wss protocols

Najvažnije metode i događaji:

**Socket.send()**

Socket.onopen()

Socket.onerror()

Socket.close()

**Socket.onmessage()**

Socket.onclose()

```
// Create WebSocket connection.  
const socket = new WebSocket('ws://localhost:8080');
```

```
// Connection opened  
socket.addEventListener('open', function (event) {  
    socket.send('Hello Server!');  
});
```

```
// Listen for messages  
socket.addEventListener('message', function (event) {  
    console.log('Message from server ', event.data);  
});
```



# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
  function WebSocketTest() {

    if ("WebSocket" in window) {
      alert("WebSocket is supported by your Browser!");

      // Let us open a web socket
      var ws = new WebSocket("ws://localhost:9998/echo");

      ws.onopen = function() {

        // Web Socket is connected, send data using send()
        ws.send("Message to send");
        alert("Message is sent...");
      };

      ws.onmessage = function (evt) {
        var received_msg = evt.data;
        alert("Message is received...");
      };

      ws.onclose = function() {

        // websocket is closed.
        alert("Connection is closed...");
      };
    } else {

      // The browser doesn't support WebSocket
      alert("WebSocket NOT supported by your Browser!");
    }
  }
</script>
```

# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
  function WebSocketTest() {

    if ("WebSocket" in window) {
      alert("WebSocket is supported by your Browser!");

      // Let us open a web socket
      var ws = new WebSocket("ws://localhost:9998/echo");

      ws.onopen = function() {

        // Web Socket is connected, send data using send()
        ws.send("Message to send");
        alert("Message is sent...");
      };

      ws.onmessage = function (evt) {
        var received_msg = evt.data;
        alert("Message is received...");
      };

      ws.onclose = function() {

        // websocket is closed.
        alert("Connection is closed...");
      };
    } else {

      // The browser doesn't support WebSocket
      alert("WebSocket NOT supported by your Browser!");
    }
  }
</script>
```

# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
  function WebSocketTest() {
```

```
    if ("WebSocket" in window) {
      alert("WebSocket is supported by your Browser!");
```

```
    // Let us open a web
    var ws = new WebSocket
```

```
    ws.onopen = function(
```

```
      // Web Socket is c
      ws.send("Message t
      alert("Message is
    };
```

```
    ws.onmessage = function (evt) {
```

```
      var rece
      alert("M
```

```
    };
```

```
    ws.onclose
```

```
      // webso
      alert("C
```

```
    };
```

```
  } else {
```

```
    // The brow
    alert("WebS
```

```
  }
```

```
}
```

```
</script>
```

Za pokretanje cjelokupnog kôda ovog jednostavnog primjera potreban je web poslužitelj koji podržava WebSocket

WebSocket ekstenzija za Apache HTTP Server:  
<https://code.google.com/p/pywebsocket/>

```
<!DOCTYPE HTML>
```

```
<html>
```

```
  <head>
```

```
    <script type = "text/javascript">
```

```
    ...
```

```
    </script>
```

```
  </head>
```

```
  <body>
```

```
    <div id = "sse">
```

```
      <a href = "javascript:WebSocketTest()">Run WebSocket</a>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

# WebSocket API – cjelovit primjer u Node.js (1)



```
// Node.js socket server script
const net = require('net');
// Create a server object
const server = net.createServer((socket) => {
  socket.on('data', (data) => {
    console.log(data.toString());
  });
  socket.write('SERVER: Hello! This is server speaking.<br>');
  socket.end('SERVER: Closing connection now.<br>');
}).on('error', (err) => {
  console.error(err);
});
// Open server on port 9898
server.listen(9898, () => {
  console.log('opened server on', server.address().port);
});
```

Poslužitelj u  
Node.js

Pokreće *socket* poslužitelj na portu 9898 (*createServer*), te „sluša” za zahtjeve za povezivanjem (*server.listen*). Šalje niz znakova (*socket.write*) na svako uspješno spajanja klijenta. Nakon toga *socket* se zatvara (*socket.end*). Pokretanje i greške se logiraju.

# WebSocket API – cjelovit primjer u Node.js (2)



```
// Node.js socket client script
const net = require('net');
// Connect to a server @ port 9898
const client = net.createConnection({ port: 9898 }, () => {
  console.log('CLIENT: I connected to the server.');
```

```
  client.write('CLIENT: Hello this is client!');
});

client.on('data', (data) => {
  console.log(data.toString());
  client.end();
});

client.on('end', () => {
  console.log('CLIENT: I disconnected from the server.');
```

Klijent u  
Node.js

Klijent se pokušava spojiti na port 9898 (createConnection). Ako je povezivanje uspješno, klijent šalje niz znakova (client.write) na poslužitelja. Ako poslužitelj pošalje poruku klijentu ona se dohvaća u 'data' metodi za upravljanjem događaja (client.on('data', (data))) te se ispisuje na ekranu (console.log).

# WebSocket API – cjelovit primjer u Node.js (3)



```
// Node.js WebSocket server script
const http = require('http');
const WebSocketServer = require('websocket').server;
const server = http.createServer();
server.listen(9898);
const wsServer = new WebSocketServer({
  httpServer: server
});
wsServer.on('request', function(request) {
  const connection = request.accept(null, request.origin);
  connection.on('message', function(message) {
    console.log('Received Message:', message.utf8Data);
    connection.sendUTF('Hi this is WebSocket server!');
  });
  connection.on('close', function(reasonCode, description) {
    console.log('Client has disconnected.');
```

WebSocket  
poslužitelj u  
Node.js

Pokreće **WebSocket API** poslužitelj na portu 9898 (`server.listen`) jednake funkcionalnosti kao prethodni poslužitelj. Koristi programski paket treće strane [WebSocket NPM package](https://www.npmjs.com/package/websocket) za jednostavno kreiranje poslužitelja (`WebSocketServer`).

<https://www.pubnub.com/blog/nodejs-websocket-programming-examples/>

# WebSocket API – cjelovit primjer u Node.js (4)



```
<!DOCTYPE html>
<html>
<head>
  <title>WebSocket Playground</title>
</head>
<body>
</body>
<script>
const ws = new WebSocket('ws://localhost:9898/');
ws.onopen = function() {
  console.log('WebSocket Client Connected');
  ws.send('Hi this is web client.');
```

```
};
ws.onmessage = function(e) {
  console.log("Received: " + e.data + "");
};
</script>
</html>
```

WebSocket  
klijent u  
Node.js

Deklaracija WebSocket klijenta (WebSocket) je podržana od HTML5 web preglednika. Koristi se metoda za upravljanjem događaja onmessage za primanje poruka sa poslužitelja.