

# STANDARDNI KORISNIČKI INTERFEJSI

Predavanje broj: 12

Nastavna jedinica: JavaScript

Nastavne teme:

Interfejs Web Storage. Session Storage. Local Storage. Interfejs Geolocation (očitanje podatka, sprega sa Google map, geokonfiguracija). Google map. Interfejs Web Messaging (slanje poruke izvora ka odredištu, komuniciranje sa određenim izvorima i odredištima). Interfejs Web Worker, SharedWorker.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

J. D. Gauchat, "Integrirane tehnologije za izradu WEB strana", Mikroknjiga, Beograd, 2014.

W3C Tutorials, Internet, 2014

# Interfejs: Web Storage

- Web Storage omogućuje da se podaci snime (ključ-vrednost) na korisnikov disk i da se kasnije upotrebe (cookie smo koristili za smeštanje male količina podataka).
- Razlikuju se:
  - **sessionStorage**,
    - skladištenje podatka koji su dostupni samo tokom trajanja sesije.
    - predstavlja zamenu za cookies
    - ovaj sistem koristi određeni prozor ili karticu (dok cookie referencira browser)
      - podaci koje napravi sessionStorage su dostupni dok se ne zatvori prozor (kartica) odgovarajuće sesije
      - cookie su dostupni dok je otvoren bilo koji prozor browser-a (za vreme definisanog života cookie-a)
  - **localStorage**,
    - radi kao storage system u tradicionalnim aplikacijama za PC, dakle, podaci se trajno skladište i uvek su dostupni iz aplikacije koja ih je napravila.
- Oba sistema rade sa istim interfejsom.
- Sledi primer u kome se realizuje skladištenje podatka korišćenjem sistema sessionStorage.

# Interfejs: Web Storage, sessionStorage

```
<!DOCTYPE html>
<html lang="en">
<head>    <meta charset="utf-8">    <title>Web Storage API</title>
    <link rel="stylesheet" href="storage.css">
    <script src="storage.js"></script>
</head>
<body>
    <section id="formbox">
        <form name="form">
            <label for="keyword">Keyword: </label><br>
            <input type="text" name="keyword" id="keyword"><br>
            <label for="text">Value: </label><br>
            <textarea name="text" id="text"></textarea><br>
            <input type="button" id="save" value="Save">
        </form>
    </section>
    <section id="databox">
        No Information available
    </section>
</body>
</html>
```

# Interfejs: Web Storage, sessionStorage

- Stilovi za okvire, storage.css:

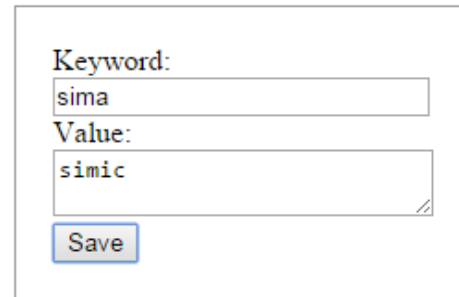
```
#formbox{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#databox{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#keyword, #text{
  width: 200px;
}
#databox > div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

# Interfejs: Web Storage, sessionStorage

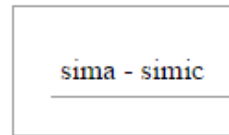
- Skladištenje i preuzimanje poslednje memorisane stavke:

```
function initiate(){
    var button = document.getElementById('save');
    button.addEventListener('click', newitem);
}
function newitem(){
    var keyword = document.getElementById('keyword').value;
    var value = document.getElementById('text').value;
    sessionStorage.setItem(keyword, value);
    //ili sessionStorage[keyword] = value;
    show(keyword);
}

function show(keyword){
    var databox = document.getElementById('databox');
    var value = sessionStorage.getItem(keyword);
    //ili var value = sessionStorage[keyword];
    databox.innerHTML = '<div>' + keyword + ' - ' + value +
    '</div>';
}
addEventListener('load', initiate);
```



Keyword:  
sima  
Value:  
simic  
Save



sima - simic

# Interfejs: Web Storage, sessionStorage

- Modifikacija storage.js koda tako da je omogućeno: definisanje, skladištenje i brisanje stavke, te preuzimanje (brisanje) liste memorisanih stavki.

```
function initiate(){
    var button = document.getElementById('save');
    button.addEventListener('click', newitem);
    show();
}
function newitem(){
    var keyword = document.getElementById('keyword').value;
    var value = document.getElementById('text').value;
    sessionStorage.setItem(keyword, value);
    document.getElementById('keyword').value = '';
    document.getElementById('text').value = '';
    show();
}
function show(){
    var databox = document.getElementById('databox');
    databox.innerHTML = '<div><input type="button"
        onclick="removeAll()" value="Erase Everything"></div>';
```

# Interfejs: Web Storage, sessionStorage

```
for(var f = 0; f < sessionStorage.length; f++){
    var keyword = sessionStorage.key(f);
    var value    = sessionStorage.getItem(keyword);
    databox.innerHTML += '<div>' + keyword + ' - ' + value +
    '<br><input type="button" onclick="removeItem(' + keyword + ')"
    value="Remove"></div>';
}
}
function removeItem(keyword){
    if(confirm('Are you sure?')){
        sessionStorage.removeItem(keyword);
        show();
    }
}
function removeAll(){
    if(confirm('Are you sure?')){
        sessionStorage.clear();
        show();
    }
}
addEventListener('load', initiate);
```

# Interfejs: Web Storage, sessionStorage

Keyword:  
nada

Value:  
nadic

Save

Erase Everything

pera - peric

Remove

sima - simic

Rem

JavaScript

Are you sure?

OK Cancel

dodavanje i  
brisanje  
stavke

Keyword:  
nada

Value:  
nadic

Save

Erase Everything

sima - simic

Remove

Keyword:

Value:

Save

Erase Everything

nada - nadic

Remove

sima - simic

Remove

JavaScript

Are you sure?

OK Cancel

brisanje svih  
stavki



# Interfejs: Web Storage, localStorage

- Pomoću sistema localStorage moguće je skladištenje veće količine podataka.
  - snimljena stavka će biti zadržana i nakon što se browser zatvori.

```
function initiate(){
    var button = document.getElementById('save');
    button.addEventListener('click', newitem);
    addEventListener('storage', show); show();
}
function newitem(){
    var keyword = document.getElementById('keyword').value;
    var value = document.getElementById('text').value;
    localStorage.setItem(keyword, value); show();
    document.getElementById('keyword').value = '';
    document.getElementById('text').value = '';
}
function show(){
    var databox = document.getElementById('databox'); databox.innerHTML = '';
    for(var f = 0; f < localStorage.length; f++){
        var keyword = localStorage.key(f);
        var value = localStorage.getItem(keyword);
        databox.innerHTML += '<div>' + keyword + ' - ' + value + '</div>';
    }
}
addEventListener('load', initiate);
```

# Interfejs: Web Storage

- Provera podržanosti localStorage-a:

```
function localStorageSupported() {  
  try {  
    return "localStorage" in window && window["localStorage"] !== null;  
  } catch (e) {  
    return false;  
  }  
}
```

- Snimanje podataka u sistemStorage (kao i u sessionStorage):

```
localStorage.setItem("key", "value");  
sessionStorage.setItem("foo", 3.14);  
localStorage.setItem("bar", true);  
sessionStorage.setItem("baz", JSON.stringify(object));
```

- Provera upisa u localStorage:

```
try {  
  localStorage.setItem("key", "value");  
} catch (e) {  
  alert("Exceeded Storage Quota!"); //npr. 5MB  
}
```

# Interfejs: Web Storage

- Čitanje podatka iz system storage-a:

```
var string    = localStorage.getItem("key");  
var number    = sessionStorage.getItem("foo");  
var boolean   = localStorage.getItem("bar");  
var object    = JSON.parse(sessionStorage.getItem("baz"));
```

ili:

```
var string    = localStorage.key;  
var number    = sessionStorage.foo;  
var boolean   = localStorage["bar"];  
var object    = JSON.parse(sessionStorage["baz"]);
```

- Iteracija nad snimljenim podacima:

```
for (var i = 0; i < localStorage.length; i++) {  
  var key = localStorage.key(i);  
  var value = localStorage.getItem(key);  
  // isto i za sessionStorage  
  ...}
```

- Brisanje snimljenih podataka:

```
localStorage.removeItem("key");  
sessionStorage.removeItem("foo");  
localStorage.removeItem("bar");  
sessionStorage.removeItem("baz");
```

# Interfejs: Web Storage

- Brisanje snimljenih podataka:

```
delete localStorage.key;  
delete sessionStorage.foo;  
delete localStorage["bar"];  
delete sessionStorage["baz"];  
localStorage.clear(); //brisanje svih članova  
sessionStorage.clear(); //brisanje svih članova
```

- Događaj storage:

```
window.addEventListener("storage", function(event) {  
    var key          = event.key;          //"kljuc"  
    var newValue     = event.newValue;    //"56"  
    var oldValue     = event.oldValue;    //null  
    var url          = event.url;         //"file:///C:/locstor.html"  
    var storageArea = event.storageArea; //[object Storage]  
    // Storage { kljuc: "56", joe: "10", length: 2 }  
    // ... obrada dogadjaja ...  
});
```

- Napomena:

- Local storage postoji dok se isti eksplicitno ne obriše ili dok se ne obriše browser-ov cache.

# Interfejs: Web Storage

- **Session storage** postoji dok se isti eksplicitno ne obriše ili dok se ne zatvori stranica koja na njega referencira.
- Podaci snimljeni u jednom browser-u nisu dostupni iz drugog browser-a.
  - Npr. podaci snimljeni u Chrome-u ne vide se u Firefox-u.
- Objekte bi trebalo snimati kao JSON stringove.
- Iz sigurnosnih razloga ne snimati osetljive podatke
  - naročito ne u localStorage.
- Snimanje podataka u system storage aktivira događaj “**storage**”.
  - U Chrome-u, ako je stranica file:///... neće biti aktiviran događaj “storage”.
    - Dakle trebalo bi da stranica bude na "domenu"
    - Koristite Mozilla-u za testiranje na stranici file:///...

# Interfejs: localStorage

```
<!DOCTYPE html>
<html>
<head>
<title>Storage</title>
<meta charset="UTF-8" />
<script>
```

```
    window.addEventListener("load", function(event) {
    var key      = document.getElementById("key");
    var value    = document.getElementById("value");
    var add      = document.getElementById("add");
    var remove   = document.getElementById("remove");
    var clear    = document.getElementById("clear");
    var content  = document.getElementById("content");
    add.addEventListener("click", function(event) {
        if (key.value !== "") {
            try {
                localStorage.setItem(key.value, value.value);
            } catch (e) { alert("Exceeded Storage Quota!"); }
            refreshContents();
        }
    });
```

Key:

Value:

Contents of Local Storage:

'KLjuc1' = '001'

# Interfejs: localStorage

```
remove.addEventListener("click", function(event) {
    if (key.value !== "") {
        localStorage.removeItem(key.value);
        refreshContents();
    }
});
clear.addEventListener("click", function(event) {
    localStorage.clear();
    refreshContents();
});
window.addEventListener("storage", function(event) {
    var k          = event.key;
    var newValue    = event.newValue;
    var oldValue    = event.oldValue;
    var url         = event.url;
    var storageArea = event.storageArea;
    alert("EVENT:\n" + k + "\n" + newValue + "\n" + oldValue +
        "\n" + url + "\n" + storageArea);
    refreshContents();
});
```

# Interfejs: localStorage

```
function refreshContents() {
    var str = "";
    for (var i = 0, len = localStorage.length; i < len; i++) {
        var k = localStorage.key(i);
        var v = localStorage.getItem(k);
        str += "'" + k + "' = '" + v + "'<br />";
    }
    key.value = ""; value.value = ""; content.innerHTML = str;
}
refreshContents();
});
</script>
</head><body>
    Key: <input type="text" id="key" /><br />
    Value: <input type="text" id="value" /><br />
    <input type="button" id="add" value="Add to Storage" />&nbsp;
    <input type="button" id="remove" value="Remove from Storage"
    />&nbsp;
    <input type="button" id="clear" value="Clear Storage" /><br />
    Contents of Local Storage:<br />
    <span id="content"></span></body></html>
```



# Interfejs: Geolocation

- Geolocation omogućuje otkrivanje geografske lokacije korisnika.
  - Koristi mrežnu triangulaciju ili GPS.
- Koriste se sledeće metode:
  - **getCurrentPosition**(f\_lokacija, f\_greska, konfiguracija)
    - koristi se za pojedinačne zahteve i ima sledeće:
      - funkciju za obradu vraćene lokacije
      - funkciju za obradu vraćenih grešaka
      - objekat koji definiše način dobijanja podataka
    - funkcija za obradu vraćene lokacije je obavezan parametar
  - **watchPosition**(lokacija, greska, konfiguracija)
    - radi slično kao metoda setInterval, automatski i periodično otkriva nove lokacije i preuzima podatke o njima.
    - metoda watchPosition vraća vrednost koja se može skladištiti u promenljivoj
  - **clearWatch**(id)
    - radi slično kao clearInterval, prekida nadzor geolokacije gde je id identifikator koji vraća metoda watchPosition

# Interfejs: Geolocation

- Sledi primer gde se klikom na dugme prikazuju podaci preuzeti pomoću sistema za geolociranje.

```
<!DOCTYPE html>
<html lang="en">
<head>  <meta charset="utf-8">  <title>Geolocation</title>
  <script src="geolocation.js"></script>
</head>
<body>
  <section id="location">
    <input type="button" id="getlocation" value="Get my location">
  </section>
</body>
</html>
```

- Za omogućavanje geolokacije u Mozilli
  - about:config
    - geo.enabled

# Interfejs: Geolocation, getCurrentPosition(lokacija)

- Povratna funkcija (u primeru showInfo) prihvata objekat tipa **Position** koji ima svojstva:
  - **coords** (sa svojstvima: latitude, longitude, altitude[m], accuracy[m], altitudeAccuracy[m], heading [°], speed[m/s])
  - **timestamp** (vreme kada su podaci dobijeni).

```
function initiate(){
    var get = document.getElementById('getlocation');
    get.addEventListener('click', getlocation);
}
function getlocation(){
    navigator.geolocation.getCurrentPosition(showinfo);
}
function showinfo(position){
    var location = document.getElementById('location');
    var data = '';
    data += 'Latitude: ' + position.coords.latitude + '<br>';
    data += 'Longitude: ' + position.coords.longitude + '<br>';
    data += 'Accuracy: ' + position.coords.accuracy + '[m]<br>';
    location.innerHTML = data;
}
addEventListener('load', initiate);
```

# Interfejs: Geolocation, getCurrentPosition(lokalacija, greska)

- Metoda **getCurrentPosition** vraća objekat tipa **PositionError** ako se otkrije greška. Ovaj objekat ima dva svojstva:
  - **code**, vrednost greške
    - **UNKNOWN\_ERROR**, vrednost **0**
    - **PERMISSION\_DENIED**, vrednost **1**, korisnik ne dozvoljava da API Geolocation pristupi njegovim podacima o lokaciji.
    - **POSITION\_UNAVAILABLE**, vrednost **2**, ne može se odrediti pozicija uređaja.
    - **TIMEOUT**, vrednost **3**, pozicija se ne može odrediti u vremenskom periodu deklarisanom u konfiguraciji geolokacije.
  - **message**, opis greške.

- U prethodnom kodu bila bi izmena:

```
function getlocation(){  
    navigator.geolocation.getCurrentPosition(showinfo, showerror);  
}
```

i dodatak:

```
function showerror(error){  
    alert('Error: ' + error.code + ' ' + error.message); }  
//moglo bi i npr. if(error.PERMISSION_DENIED===error.code)...
```

- Treći mogući parametar predstavlja objekat sa najviše tri svojstva koja određuju konfiguraciju u smislu načina dobijanja podataka:
  - **enableHighAccuracy**,
    - ovo logičko svojstvo informiše sistem o tome da se traži najpreciznija moguća lokacija.
      - browser će pokušati da dobije podatke od GPS-a kako bi se izračunala precizna lokacija.
      - pošto su ovi sistemi veoma zahtevni sa stanovišta korišćenja resursa i njihovu ulogu treba ograničiti na posebne okolnosti, zato je podrazumevana vrednost ovog svojstva false.
  - **timeout**,
    - ovo svojstvo pokazuje maksimalno vreme trajanja operacije, tako da ako se podaci ne dobiju u okviru vremenskog ograničenja, vraća se greška TIMEOUT. Podrazumevana vrednost je trajno čekanje.
    - vrednost ovog svojstva izražava se u milisekundama.
  - **maximumAge**,
    - pošto se prethodne pozicije keširaju u sistemu, može se postaviti vreme u milisekundama u okviru koga se zbog bržeg odziva uzima poslednja keširana vrednost, inače se uzima nova lokacija od sistema.

# Konfigurisanje geolokacijskog sistema

```
function initiate(){
    var get = document.getElementById('getlocation');
    get.addEventListener('click', getlocation);
}
function getlocation(){
    var geoconfig =
    { enableHighAccuracy: true, timeout: 10000, maximumAge: 60000 };
    navigator.geolocation.getCurrentPosition(showinfo, showerror,
                                             geoconfig);
}
function showinfo(position){
    var location = document.getElementById('location'); var data='';
    data += 'Latitude: ' + position.coords.latitude + '<br>';
    data += 'Longitude: ' + position.coords.longitude + '<br>';
    data += 'Accuracy: ' + position.coords.accuracy + '[m]<br>';
    location.innerHTML = data;
}
function showerror(error){
    alert('Error:'+error.code+' '+error.message);
}
addEventListener('load', initiate);
```

- Kao i metoda `getCurrentPosition` i metoda `watchPosition` izračunava lokaciju uređaja.
  - Ova metoda izračunava podatke *kada god se lokacija promeni*.
  - Ovaj proces se otkazuje metodom `clearWatch`.
  - Da bi se probala ova metoda potrebno je da se aplikacija instalira na mobilni uređaj i onda bi promenom pozicije došlo do promene podatka o lokaciji.

```
function initiate(){
    var get = document.getElementById('getlocation');
    get.addEventListener('click', getLocation);
}
function getLocation(){
    var geoconfig = {
        enableHighAccuracy: true,
        maximumAge: 60000
    };
    control = navigator.geolocation.watchPosition(showInfo,
                                                    showError,
                                                    geoconfig);
}
```

```
function showInfo(position){
    var location = document.getElementById('location');
    var data = '';
    data += 'Latitude: ' + position.coords.latitude + '<br>';
    data += 'Longitude: ' + position.coords.longitude + '<br>';
    data += 'Accuracy: ' + position.coords.accuracy + 'mts.<br>';
    location.innerHTML = data;
}
function showError(error){
    alert('Error: ' + error.code + ' ' + error.message);
}
addEventListener('load', initiate);
```

- Atribut maximumAge određuje koliko će se često informacija o promeni lokacije slati metodi showInfo.
  - U datom primeru, ako se nova lokacija evidentira 1 minut nakon što je evidentirana prethodna lokacija ona će biti prikazana, inače se funkcija showInfo ne poziva.
- Povratna vrednost metode watchPosition se čuva u promenljivoj (u primeru control) i predstavlja id ove operacije.



# Geolocation + Google Maps

- U funkciji showInfo će vrednost objekta Position biti dodata Google URL adresi, a onda se adresa umeće kao izvor elementa <img> da bi se prikazala vraćena slika.

```
function initiate(){
    var get = document.getElementById('getlocation');
    get.addEventListener('click', getlocation);
}
function getlocation(){
    navigator.geolocation.getCurrentPosition(showinfo, showerror);
}
function showinfo(position){
    var location = document.getElementById('location');
    var mapurl = http://maps.google.com/maps/api/staticmap?
        sensor=false&center=' +
        position.coords.latitude + ',' + position.coords.longitude
        + '&zoom=12&size=400x400&markers=' +
        position.coords.latitude + ',' + position.coords.longitude;
    location.innerHTML = '';
}
function showerror(error){
    alert('Error:' + error.code + ' ' + error.message);
}
addEventListener('load', initiate);
```

# Geolocation + Google Maps



# Prikaz mape preko Google APIa

```
<!DOCTYPE html>
<html><body>
<div id="map" style="width:100%;height:500px"></div>

<script>
function myMap() {
  var mapCanvas = document.getElementById("map");
  var myCenter = new google.maps.LatLng(51.508742,-0.120850);
  var mapOptions = {center: myCenter, zoom: 5};
  var map = new google.maps.Map(mapCanvas,mapOptions);
  var marker = new google.maps.Marker({
    position: myCenter,
    animation: google.maps.Animation.BOUNCE
  });
  marker.setMap(map);
}
</script>

<script
src="https://maps.googleapis.com/maps/api/js?callback=myMap"></script>

</body>
</html>
```



# Interfejs: Web Messaging

- Omogućuje slanje poruka iz jednog dokumenta u drugi
  - razmena poruka između dokumenata (cross-document messaging)
  - poruka se šalje iz jednog a obrađuje u drugom dokumentu
- Slanje poruke obavlja metoda:
  - **postMessage**(poruka, odredište)
    - Šalje poruku u drugi dokument
      - parametar poruka je znakovni niz koji se prenosi
      - parametar odredište je domen odredišnog dokumenta (npr. ime servera, priključak)
        - » simbol **\*** označava da je odredište bilo koji dokument
        - » simbol **/** označava da se odredište isto kao i izvor
    - Metoda za komunikaciju je asinhrona.
    - Pripada objektu **window**.
- Za osluškivanje poruka poslatih u odredišni dokument koristi se događaj **'message'**.

# Interfejs: Web Messaging

- Svojstva objekta kreiranog događajem **message**:
  - **data**,       sadržaj poruke
  - **origin**,     izvor dokumenta koji je poslao poruku (mogli bismo da vratimo poruku pošiljaocu)
  - **source**,     objekat za identifikovanje izvora poruke

```
<!DOCTYPE html>
<html lang="en"><head>  <meta charset="utf-8">  <title>CDM</title>
  <link rel="stylesheet" href="messaging.css">
  <script src="messaging.js"></script>
</head>
<body>
  <section id="formbox">
    <form name="form">
      <label for="name">Your name: </label>
      <input type="text" name="name" id="name" required>
      <input type="button" id="button" value="Send">
    </form>
  </section>
  <section id="databox">    <iframe id="iframe" src="iframe.html"
width="500" height="350"></iframe>  </section>
</body></html>
```

# Interfejs: Web Messaging

- Stilizovanje, datoteka messaging.css:

```
#formbox{
  float: left;   padding: 20px;   border: 1px solid #999999;
}
#databox{
  float: left;   width: 500px;   margin-left: 20px;
  padding: 20px;   border: 1px solid #999999;
}
```

- Datoteka messaging.js:

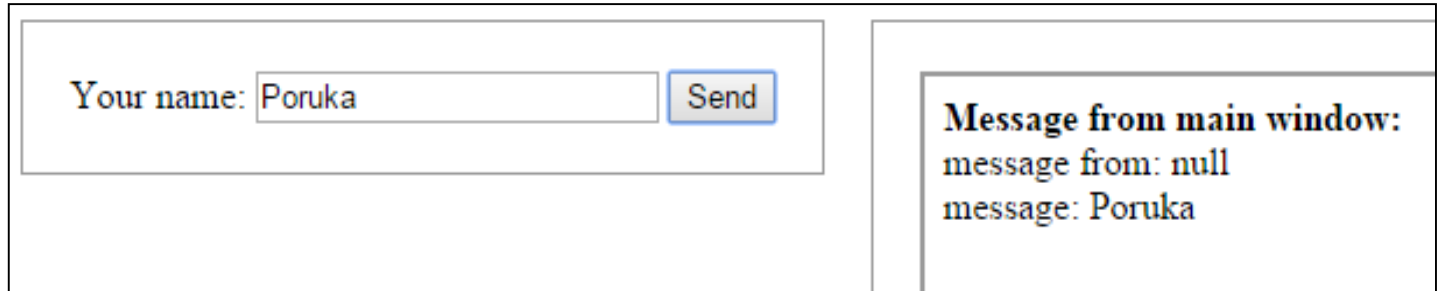
```
function initiate(){
  var button = document.getElementById('button');
  button.addEventListener('click', send);
}
function send(){
  var name = document.getElementById('name').value;
  var iframe = document.getElementById('iframe');

  iframe.contentWindow.postMessage(name, '*');//za sve d. u iframe
}
addEventListener('load', initiate);
```

# Interfejs: Web Messaging

- Datoteka iframe.html

```
<!DOCTYPE html>
<html lang="en"><head><meta charset="utf-8"> <title>iframe</title>
  <script src="iframe.js"></script>
</head>
<body>  <section>
  <div><b>Message from main window:</b></div>
  <div id="databox"></div>
</section>
</body>
</html>
```



Your name:

**Message from main window:**  
message from: null  
message: Poruka

- Datoteka iframe.js

```
function initiate(){  addEventListener('message', receiver); }
function receiver(e){
  var databox = document.getElementById('databox');
  databox.innerHTML  = 'message from: ' + e.origin + '<br>';
  databox.innerHTML += 'message: ' + e.data;
}
addEventListener('load', initiate);
```

# Komuniciranje sa određenim izvorima i odredištima

```
<!DOCTYPE html>
<html lang="en">
<head>  <meta charset="utf-8">
        <title>Cross Document Messaging</title>
        <link rel="stylesheet" href="messaging.css">
        <script src="messaging.js"></script>
</head>
<body>
  <section id="formbox">
    <form name="form">
      <label for="name">Your name: </label>
      <input type="text" name="name" id="name" required>
      <input type="button" id="button" value="Send">
    </form>
  </section>
  <section id="databox">
    <iframe id="iframe" src="http://www.domain2.com/iframe.html"
width="500" height="350"></iframe>
  </section>
</body>
</html>
```



# Komuniciranje sa određenim izvorima i odredištima

Datoteka messaging.js

```
function initiate(){
    var button = document.getElementById('button');
    button.addEventListener('click', send);
    addEventListener('message', receiver);
}
function send(){
    var name = document.getElementById('name').value;
    var iframe = document.getElementById('iframe');
    iframe.contentWindow.postMessage(name,
                                    'http://www.domain2.com');
}
function receiver(e){
    if(e.origin == 'http://www.domain2.com'){
        document.getElementById('name').value = e.data;
    }
}
addEventListener('load', initiate);
```

# Komuniciranje sa određenim izvorima i odredištima

Datoteka iframe.js

```
function initiate(){
  addEventListener('message', receiver);
}
function receiver(e){
  var databox = document.getElementById('databox');
  if(e.origin == 'http://www.domain1.com'){
    databox.innerHTML = 'valid message: ' + e.data;
    e.source.postMessage('message received', e.origin);
  }
  else{
    databox.innerHTML = 'invalid origin';
  }
}
addEventListener('load', initiate);
```

# Interfejs: Web workers

- Kada se skript izvršava u HTML stranici, stranica ne reaguje dok se skript ne završi.
- Interfejs web worker omogućuje izvršavanje JavaScript-a u pozadini nezavisno od ostalih skriptova bez uticaja na performanse stranice.
  - Za vreme ovakvog izvršavanja skripta može se normalno komunicirati sa stranicom (reagovanje na klik, selekciju ... ).

- Pre kreiranja web worker-a mora se proveriti da li browser podržava web worker-e:

```
if(typeof(Worker) !== "undefined") {  
    // OK! Web worker je podrzan!  
    // ...  
} else {  
    // NOK! Web worker nije podrzan.  
}
```

- Pre komunikacije sa web worker-om potreban je objekat koji pokazuje na datoteku sa kodom web radnika:
  - **Worker**(scriptURL) - konstruktor koji daje objekat tipa Worker, a scriptURL je datoteka sa kodom koji će se obrađivati u pozadini

# Interfejs: Web workers

- Potrebno je proveriti da li worker već postoji te ako ne postoji kreirati novi web worker objekat koji pokazuje na web worker datoteku ("demo\_workers.js") koja će biti pozvana iz HTML stranice:

```
if(typeof(w) == "undefined") {  
    w = new Worker("demo_workers.js");  
}
```

- Sada se mogu slati poruke ka workeru i primiti poruke od istog.
- Poruka koja se iz glavnog koda šalje web worker-u je informacija koju treba obraditi, a poruka koju vraća worker (radnik) je rezultat te obrade.
  - **postMessage**(poruka)
    - poruka koja se šalje (prethodno poglavlje).
  - **message**
    - događaj koji osluškuje poruke koje se šalju (prethodno poglavlje) i kreira objekat sa svojstvom data koji se odnosi na sadržaj poruke.
    - workeru se dodaje osluškivač na događaj "**onmessage**"

```
w.onmessage = function(event){  
    document.getElementById("result").innerHTML = event.data;  
};
```

# Interfejs: Web workers

- Kada web worker pošalje poruku izvršava se osluškivač događaja pri čemu se podaci web workera smeštaju u svojstvo event.data, a oni u selektovani element.
- Otkazivanje rada web worker-a
  - Kada je kreiran objekat web worker on nastavlja da osluškuje poruke (čak i nakon što je eksterni skript izvršen) sve dok se ne otkaže rad web worker-a.
  - Za terminiranje web worker-a i oslobađanje browser-vih i računarskih resursa koriste se metode:  
worker\_objekat.terminate();
    - Metoda terminate otkazuje web worker-a u glavnom kodu.
  - close():
    - Metoda close otkazuje web worker-a u samom kodu radnika.
  - Ako se worker otkaže, prekidaju se svi procesi koji se u tom trenutku izvršavaju, a svi zadaci u petlji događaja se odbacuju.
- Ponovno korišćenje web worker-a
  - Ako se, nakon što je web worker terminiran, postavi da je promenljiva worker undefined, omogućeno je ponovno korišćenje koda.  
w = undefined;

# Interfejs: Web workers

- Brojač koji radi u pozadini:

```
<!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, your browser does not support Web Workers...";
```

```
  }
```

```
}
```

```
function stopWorker() {
```

```
  w.terminate();    w = undefined; }
```

```
</script></body></html>
```

Count numbers:

Start Worker

Stop Worker

Count numbers: 5

Start Worker

Stop Worker

# Interfejs: Web workers

- Web worker koji odbrojava i šalje vrednost svog brojača u glavni program.
- Datoteka demo\_workers.js:

```
var i = 0;
function timedCount() {
    i = i + 1;
    postMessage(i);
    setTimeout("timedCount()",500);
}
timedCount();
```

objašnjenje:

- ***timedCount()***:
  - funkcija kreiranog web workera koja prosleđuje vrednost brojača ka glavnoj stranici korišćenjem metode `postMessage()`
  - pozvana funkcija `timedCount` se aktivira metodom `setTimeout` svakih 0.5 sekundi kada se i šalje vrednost brojača glavnoj stranici

- **Napomena:**

Web workeri se koriste onda kada je potrebno obaviti poslove koji zahtevaju veliko procesorsko vreme.

# Interfejs: Web workers

- Slanje podataka, obrada u pozadini, dobijanje odgovora:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Web Workers API</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="webworkers.js"></script>
</head>
<body>
  <section id="formbox">
    <form name="form">
      <label for="name">Name: </label><br>
      <input type="text" name="name" id="name"><br>
      <input type="button" id="button" value="Send">
    </form>
  </section>
  <section id="databox"></section>
</body>
</html>
```



# Interfejs: Web workers

- Stilizovanje glavne stranice:

```
#formbox{float: left; padding: 20px;border:1px solid #999999;}  
#databox{float: left; width: 500px; margin-left: 20px;  
padding: 20px; border: 1px solid #999999;}
```

- Web webworkers.js kod:

```
var worker, databox;  
function initiate()  
{  
    databox = document.getElementById('databox');  
    var button = document.getElementById('button');  
    button.addEventListener('click', send);  
    worker = new Worker('worker.js');  
    worker.addEventListener('message', received);  
}  
function send()  
{  
    var name = document.getElementById('name').value;  
    worker.postMessage(name);  
}  
function received(e){  
    databox.innerHTML = e.data;  
}  
addEventListener('load', initiate);
```

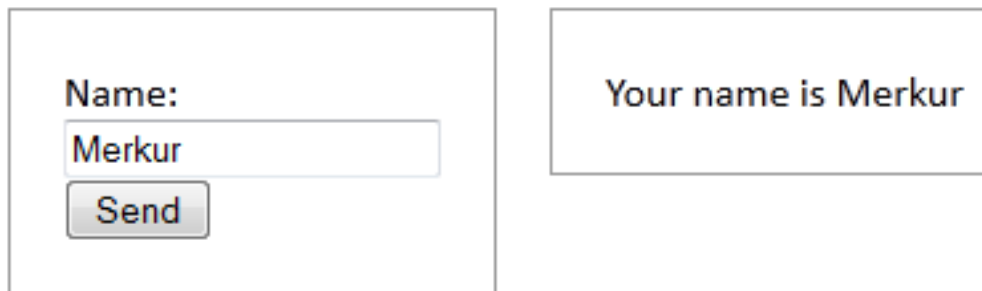
# Interfejs: Web workers

- Datoteka web workera worker.js  
`addEventListener('message', received);`

```
function received(e){  
  var answer = 'Your name is '+e.data;  
  postMessage(answer);  
}
```

worker radi sa prosleđenim podacima i šalje odgovor glavnom programu čijim elementima NE može da pristupa jer je u pitanju zatvoren kod.

**Napomena:** Paziti kod Chrome-a, kao što je ranije rečeno, ne radi u lokalu.  
Koristiti Mozilla.



The image shows a simple web interface. On the left, there is a form with a label 'Name:' followed by a text input field containing the word 'Merkur'. Below the input field is a 'Send' button. To the right of the form is a rectangular box containing the text 'Your name is Merkur', which is the response from the web worker.

# Interfejs: Web workers

- Obrada greške

```
var worker, databox;
function initiate(){
    databox = document.getElementById('databox');
    var button = document.getElementById('button');
    button.addEventListener('click', send);

    worker = new Worker('worker.js');
    worker.addEventListener('error', error);
}
function send(){
    var name = document.getElementById('name').value;
    worker.postMessage(name);
}
function error(e){
    databox.innerHTML = 'ERROR: ' + e.message + '<br>';
    databox.innerHTML += 'Filename: ' + e.filename + '<br>';
    databox.innerHTML += 'Line Number: ' + e.lineno;
}
addEventListener('load', initiate);
```

# Interfejs: Web workers

- Sada u web workeru može da se generiše greška, npr:

```
addEventListener('message', received);
```

```
function received(e){  
  test();  
}
```

Name:

ERROR: ReferenceError: test is not defined  
Filename: file:///C:/\_\_Projects/worker2.js  
Line Number: 4

- Web workeru se dodaje obrađivač greške tako da ako se desi greška u kodu radnika biće aktiviran objekat workera u glavnom kodu
  - objekat koji se generiše pri javljanju greške sadrži svojstva
    - **message** - poruka o grešci
    - **filename** - datoteka u kojoj je nastala greška
    - **lineno** - broj reda u kome je nastala greška

# Interfejs: Web workers

- Terminiranje web workera iz glavnog koda:

```
var worker, databox;
function initiate(){
    databox = document.getElementById('databox');
    var button = document.getElementById('button');
    button.addEventListener('click', send);
    worker = new Worker('worker.js');
    worker.addEventListener('message', received);
}
function send(){
    var name = document.getElementById('name').value;
    if(name == 'close1'){
        worker.terminate(); //S A M O je terminiran, sta nedostaje?
        databox.innerHTML = 'Worker Terminated';
    }else{
        worker.postMessage(name);
    }
}
function received(e){
    databox.innerHTML = e.data;}
addEventListener('load', initiate);
```

# Interfejs: Web workers

- Terminiranje web worker-a iz samog koda radnika:

```
addEventListener('message', received);
function received(e){
  if(e.data == 'close2'){
    postMessage('Worker Closed');
    close();
  }
  else{
    var answer = 'Your name is ' + e.data;
    postMessage(answer);
  }
}
```

- Učitavanje spoljne JavaScript datoteke u kod radnika (web worker-a) pri čemu će tek pri završenom učitavanju kod radnika biti spreman za izvršavanje:

```
importScripts('morecodes.js');//moze i vise skriptova del. je ,
addEventListener('message', received);
function received(e){
  test(); //npr. test je u morecodes.js
}
```

# SharedWorker

- Worker je namenski radnik (**dedicated worker**) i on se samo odazivao glavnom kodu u kome je definisan.
- Deljeni radnik (**shared worker**) se odaziva većem broju dokumenata iz istog izvora, novi konstruktor izgleda kao što sledi:  
**SharedWorker(skriptURL)**
  - skriptURL - datoteka deljenog radnika
  - SharedWorker vraća objekat tipa SharedWorker kod koga je postavljen član port (**MessagePort**) s vrednošću priključka preko koga će se ostvarivati veza sa deljenim radnikom.
- Veze se ostvaruju preko priključaka koji se mogu snimiti unutar web worker-a za buduće referenciranje.
- Nova svojstva i metode su:
  - **port** - kada se napravi SharedWorker objekat onda se definiše nov priključak i dodeljuje svojstvu port
  - **connect** - događaj za otkrivanje novih veza deljenog radnika (aktivira se kada dokument počne vezu sa deljenim radnikom)
  - **start** - metoda dostupna za objekte MessagePort koji se vraćaju prilikom definisanja deljenog radnika, a namenjena je za započinjanje otpremanja poruka primljenih preko priključaka.

# SharedWorker

- Html dokument koji testira deljene radnike

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Web Workers</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="sharedwebworkers.js"></script>
</head>
<body>
  <section id="formbox">
    <form name="form">
      <label for="name">Name: </label>
      <input type="text" name="name" id="name">
      <input type="button" id="button" value="Send">
    </form>
  </section>
  <section id="databox">
    <iframe id="iframe" src="iframe.html" width="500"
                                              height="350"></iframe>
  </section></body></html>
```



# SharedWorker

- Dokument `sharedwebworkers.js`

```
var shworker;
function initiate(){
    var button = document.getElementById('button');
    button.addEventListener('click', send);
    shworker = new SharedWorker('sharedworker.js');
    shworker.port.addEventListener('message', received);
    shworker.port.start();
}
function send(){
    var name = document.getElementById('name').value;
    shworker.port.postMessage(name);
}
function received(e){
    alert(e.data); //u glavnom
}

addEventListener('load', initiate);
```

# SharedWorker

- Html dokument `iframe.html` (na glavnoj stranici):

```
<!DOCTYPE html>
<html lang="en">
<head>  <meta charset="utf-8">  <title>iframe Window</title>
  <script src="iframe.js"></script>
</head>
<body>
  <section id="databox"></section>
</body>
</html>
```
- Dokument `iframe.js`

```
function initiate(){
  var shworker = new SharedWorker('sharedworker.js');
  shworker.port.addEventListener('message', received);
  shworker.port.start();
}
function received(e){
  var databox = document.getElementById('databox');
  databox.innerHTML = e.data;
}
addEventListener('load', initiate);
```

# SharedWorker

- Dokument `sharedworker.js`

```
var myports = [];  
addEventListener('connect', connect);  
function connect(e){  
  myports.push(e.ports[0]);  
  e.ports[0].onmessage = send;  
}  
function send(e){  
  for(var f = 0; f < myports.length; f++){  
    myports[f].postMessage('Your Name is ' + e.data);  
  }  
}
```

Name: <input type="text" value="Elvis"/> <input type="button" value="Send"/>	<div></div>
--	-------------

Name: <input type="text" value="Elvis"/> <input type="button" value="Send"/>	<div>Your Name is Elvis</div> <div><div>Your Name is Elvis</div><div><input type="button" value="OK"/></div></div>
--	--

Name: <input type="text" value="Elvis"/> <input type="button" value="Send"/>	<div>Your Name is Elvis</div>
--	-------------------------------