Local storage, session storage

Kolačići, o kojima je bilo reči u prethodnoj lekciji, dugo su bili jedini način da se određeni podaci sačuvaju na klijentu, odnosno unutar web pregledača. Ipak, u prethodnoj lekciji ste mogli da pročitate da se podaci kolačića zajedno sa HTTP zahtevom automatski prosleđuju web serveru. Stoga se nameće pitanje: *Šta ukoliko želimo da na klijentu sačuvamo podatke koji neće biti dostupni serveru?*

Web aplikacije veoma često imaju potrebu da unutar web pregledača sačuvaju određene podatke koji će se koristiti isključivo na klijentu. Zbog već spomenutih osobina, kolačići nisu pogodni za obavljanje takvog posla. Upravo zbog toga su proizvođači web pregledača razvili još neke mehanizme za čuvanje klijentski specifičnih podataka. Mehanizmi koji su direktna alternativa kolačićima nazivaju se *local storage* i *session storage*. Ove dve vrste klijentskih skladišta podataka biće predmet lekcije koja je pred vama.

Web Storage API

Web Storage API je naziv za skup funkcionalnosti koje web pregledači obezbeđuju kako bi se rukovalo local storage i session storage skladištima. Osnovne osobine takvih skladišta su:

- Podaci local storage i session storage skladišta čuvaju se u formi parova ključeva i vrednosti.
- Podaci iz local storage i session storage skladišta nikada se automatski ne prosleđuju serveru.
- HTTP server ni na koji način ne može da pristupi local storage i session storage skladištima, za razliku od kolačića, gde korišćenjem HTTP zaglavlja server može da kreira kolačiće i menja osobine kolačića.
- Podacima local storage i session storage skladišta može da rukuje samo klijent – pregledač ili JavaScript kod posredstvom pregledača.
- Svako local storage i session storage skladište vezano je za jedan origin; reč je o pojmu o kome je već bilo reči u jednoj od prethodnih lekcija – origin predstavlja kombinaciju domena, protokola i porta.

Session storage kreira se za svaku instancu jednog origina i njegovi podaci se čuvaju sve dok je sesija web pregledača aktivna. To praktično znači da se zatvaranjem taba u kojem je učitana stranica sa podacima session storagea takvi podaci gube. Jedina akcija koju session storage podaci mogu da prežive jeste osvežavanje stranice. Pored toga, podaci session storagea nikada se ne dele između više stranica istog origina koje mogu biti otvorene u različitim tabovima. Ukoliko je otvoren veći broj tabova sa stranicama istog origina, svaki tab poseduje sopstvenu instancu session storagea.

Local storage deli se između svih aktivnih tabova sa stranicama istog origina. Pored toga, podaci local storagea čuvaju se čak i kada se tab ili kompletan web pregledač zatvori. Jedini način za brisanje local storage podataka je da oni budu obrisani od strane JavaScript jezika ili eksplicitno od strane web pregledača, brisanjem keša.

Centralne figure Web Storage API-a, koje omogućavaju pristup local storage i session storage skladištima i rukovanje njihovim podacima, jesu:

- svojstva `Window` objekta `Window.localStorage` i `Window.sessionStorage`, koja omogućavaju pristup konkretnim skladištima;
- objekat `Storage`, čija svojstva i metode omogućavaju dodavanje, čitanje, izmenu i uklanjanje podataka iz local storage i session storage skladišta;
- objekat `StorageEvent`, kojim se predstavljaju događaji koji se emituju prilikom promena na skladištima.

Local storage skladištu pristupa se korišćenjem svojstva `Window.localStorage`, a session storageu korišćenjem svojstva `Window.sessionStorage`. Oba svojstva čuvaju referencu na objekat identičnog tipa – `Storage`, te je stoga nastavak rada sa local storage i session storage skladištima istovetan i zasniva se na korišćenju identičnog skupa svojstava i metoda:

- `setItem(key, value)` – upisuje par ključa i vrednosti unutar skladišta;
- `getItem(key)` – čita vrednost na osnovu ključa;
- `removeItem(key)` – uklanja par ključa i vrednosti na osnovu prosleđenog ključa;
- `clear()` – briše sve podatke iz skladišta;
- `key(index)` – čita naziv ključa koji se nalazi na prosleđenoj poziciji indeksa;
- `length` – predstavlja broj zapisa unutar skladišta.

Pitanje

Funkcionalnosti za rukovanje local storage i session storage skladištima objedinjene su unutar:

- **Web Storage API-a**
- DOM API-a
- Fetch API-a
- History API-a

Objašnjenje:

Web Storage API je objedinjeni naziv za skup funkcionalnosti koje web pregledači obezbeđuju kako bi se rukovalo local storage i session storage skladištima.

Upisivanje podataka

Podaci se u local storage i session storage mogu upisati korišćenjem metode **setItem()**:

```
setItem(key, value)
```

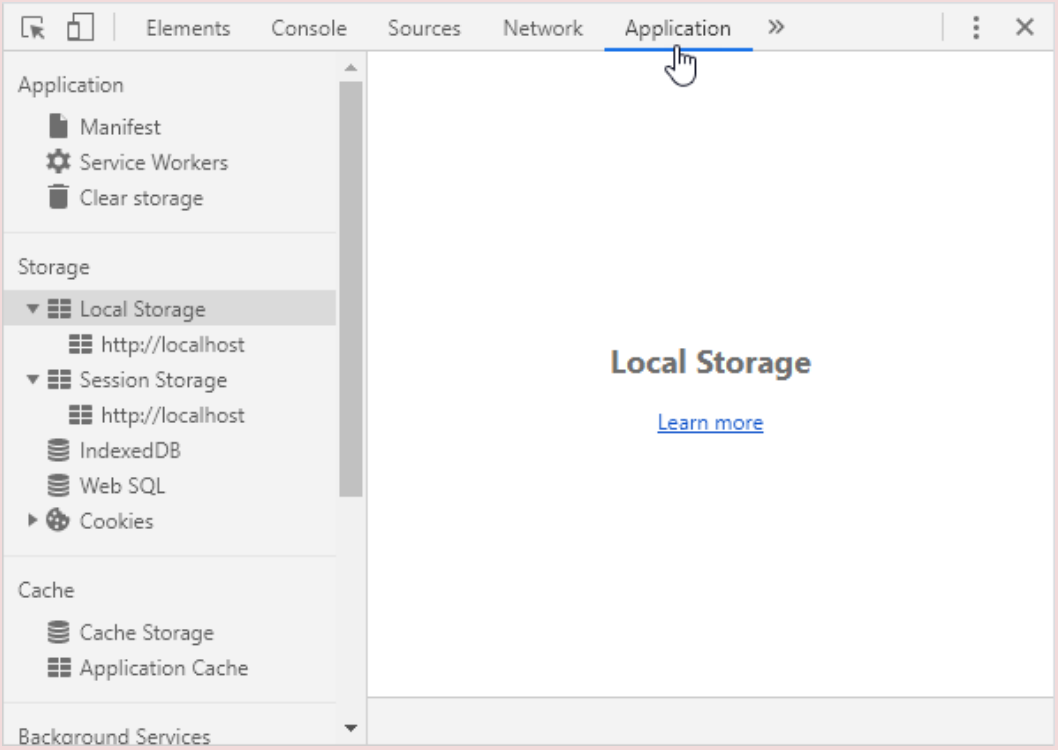
Metodi `setItem()` se prosleđuje par ključa i vrednosti:

```
sessionStorage.setItem('myDog', 'Lisa');

//or

localStorage.setItem('myDog', 'Lisa');
```

Naredbe ilustruju upisivanje identičnog para ključa i vrednosti u local storage i session storage. Drugim rečima, tako će i unutar local storagea i unutar session storagea postojati ključ `myDog` sa vrednošću `Lisa`.

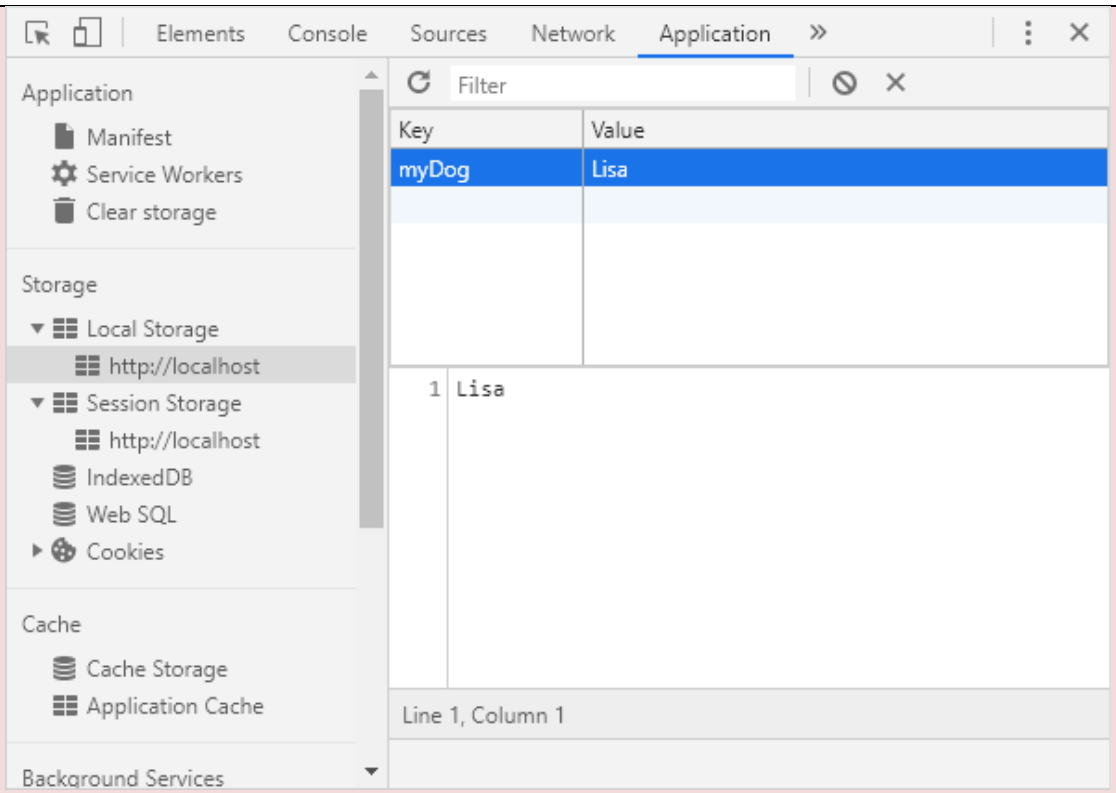
Pregled local storage i session storage skladišta u pregledaču Chrome

Stanje local storage i session storage skladišta veoma lako je moguće pregledati korišćenjem svih modernih pregledača. U Chromeu je dovoljno unutar *Developer Tools* panela odabrati *Application* tab (slika 12.1).



Slika 12.1. Pregled local storagea i session storagea u pregledaču Chrome

Sa leve strane, unutar sekcije *Storage*, mogu se videti različiti mehanizmi za čuvanje podataka unutar pregledača, među kojima i *Local Storage* i *Session Storage*. Svaki od njih poseduje i dodatnu stavku koja predstavlja konkretnu instancu skladišta dostupnu aktivnom dokumentu (u primeru sa slike je to origin – `http://localhost`). Klikom na takvu stavku dobijaju se svi zapisi upisani u skladište (slika 12.2).



Slika 12.2. Pregled local storagea i session storagea u pregledaču Chrome (2)

Klikom na odgovarajuću instancu skladišta, sa desne strane se pojavljuju svi parovi ključeva i vrednosti upisani u takvo skladište.

Na upravo prikazanom primeru upisivanja podataka u local storage i session storage možete da uvidite i osnovnu razliku između njih. Naime, dovoljno je da postavite pod komentare naredbe iz upravo prikazanog primera i onda prikazani HTML dokument otvorite u novom tabu. Moći ćete da vidite da u novom tabu nema podataka unutar session storagea, dok ih unutar local storagea ima.

Napomena

Local storage i session storage omogućavaju čuvanje podataka isključivo u `string` obliku. Štaviše, i ključevi i vrednosti koji se prosleđuju metodi `setItem()` moraju se definisati u `string` obliku.

Kada se metodi `setItem()` prosledi neka vrednost koja nije tekst, ta vrednost se automatski konvertuje u `string`:

```
localStorage.setItem('demo', 1);
```

U ovakvom slučaju, numerička vrednost `1` će biti konvertovana u tekst `'1'`.

Ipak, iako local storage i session storage omogućavaju prvenstveno čuvanje tekstualnih podataka, moguće je čuvati i podatke složenih tipova, u JSON obliku:

```
localStorage.setItem('user', JSON.stringify({name: "Ben"}));
```

Čitanje podataka

Podaci iz skladišta mogu se čitati korišćenjem metode **getItem()**:

```
getItem(key)
```

Metodi `getItem()` se prosleđuje ključ čiju vrednost želimo da pročitamo:

```
var value = localStorage.getItem('myDog');
console.log(value);

//or

var value2 = sessionStorage.getItem('myDog');
console.log(value2);
```

Pristup ključevima korišćenjem dot notacije

Čitanje i postavljanje vrednosti unutar local i session skladišta moguće je postići i korišćenjem dot notacije. Dot notacija podrazumeva korišćenje tačke, nakon koje je moguće definisati ključ, baš kao što se to radi prilikom kreiranja i čitanja vrednosti svojstava JavaScript objekata:

```
localStorage.myDog = "Lisa";

var value = localStorage.myDog;
console.log(value);
```

Ažuriranje podataka

Ažuriranje podataka unutar local i session skladišta moguće je obaviti korišćenjem metode `setItem()`, koja se koristi i za kreiranje novih zapisa:

```
localStorage.setItem('myDog', 'Ticky');
```

Na ovaj način će biti obavljeno ažuriranje vrednosti koja se nalazi pod ključem `myDog`.

Na osnovu upravo prikazanog primera, može se zaključiti da unutar local storagea i session storagea ključevi moraju biti jedinstveni. Nije moguće imati dva para ključeva i vrednosti sa identičnim ključevima.

Brisanje podataka

Brisanje zapisa iz local i session skladišta moguće je obaviti korišćenjem metode **removeItem()**:

```
removeItem(key)
```

Metodi `removeItem()` prosleđuje se ključ koji zajedno sa pripadajućom vrednošću želimo da obrišemo:

```
localStorage.removeItem("myDog");

//or

sessionStorage.removeItem("myDog");
```

Moguće je obrisati i sve podatke iz oba skladišta odjednom, korišćenjem metode **clear()**:

```
localStorage.clear();

//or

sessionStorage.clear();
```

Prolazak kroz zapise skladišta

Za prolazak kroz zapise local storagea i session storagea moguće je koristiti nekoliko pristupa. Prvi pristup podrazumeva korišćenje metode **key()**, kojom se može doći do naziva ključa na osnovnu njegovog indeksa:

```
let key = localStorage.key(0);
```

Na ovaj način, nakon izvršavanja prikazane naredbe, unutar promenljive `key` biće smešten naziv ključa prvog zapisa koji je upisan unutar local storagea. Čita se naziv ključa prvog zapisa, zato što je metodi `key()` prosleđena vrednost `0`.

Ovako dobijen ključ se može iskoristiti i za dobijanje vrednosti, njegovim prosleđivanjem metodi `getItem()`. Kompletan primer prolaska kroz sve zapise skladišta i ispisa vrednosti može da izgleda ovako:

```
localStorage.setItem('item1', 1);
localStorage.setItem('item2', 2);
localStorage.setItem('item3', 3);
localStorage.setItem('item4', 4);

for (let i = 0; i < localStorage.length; i++) {
      let key = localStorage.key(i);
      console.log(key + ": " + localStorage.getItem(key));
}
```

U primeru se prvo obavlja upisivanje četiri para ključeva i vrednosti unutar local storagea. Zatim se korišćenjem jedne for petlje prolazi kroz sve local storage zapise. Kako bi se došlo do ključeva, koristi se metoda `key()` kojoj se u svakoj iteraciji prosleđuje nova vrednost brojača `i`. Dobijeni ključevi se koriste za čitanje vrednosti, pa se tako na kraju unutar konzole dobija:

```
item2: 2
item1: 1
item3: 3
item4: 4
```

Na osnovu dobijenog ispisa može se videti još jedna značajna osobina rada sa local i session skladištima. Ona ni na koji način ne garantuju da će podaci biti upisani nekim posebnim redosledom. Iako je nešto ranije naredba za upisivanje `item1` stavke bila navedena prva, sada možemo da vidimo da se čitanjem svih vrednosti prvo dobija stavka `item2`.

Drugi način za prolazak kroz sve upisane stavke skladišta jeste korišćenje `for...of` petlje. Ipak, `for...of` se ne može direktno koristiti nad `Storage` objektom koji se dobija preko `localStorage` i `sessionStorage` svojstava, zato što `Storage` nije *iterable objekat*. Ipak, kolekcija njegovih ključeva jeste, pa se stoga može napisati ovako nešto:

```
for (let key of Object.keys(localStorage)) {
      console.log(key + ": " + localStorage.getItem(key));
}
```

U primeru se korišćenjem `for...of` petlje prolazi kroz niz koji se dobija kao povratna vrednost metode `Object.keys()`. Reč je o metodi koja od naziva sopstvenih svojstava jednog objekta kreira niz vrednosti. Unutar `for...of` petlje zatim se, kao u prethodnom primeru, korišćenjem `getItem()` metode dolazi do konkretnih vrednosti.

storage događaj

Na početku ove lekcije rečeno je da je deo Storage API-a i jedan poseban objekat kojim se predstavljaju podaci događaja do kojih dolazi nakon promena u local i session skladištima. Reč je o objektu `StorageEvent`.

`StorageEvent` objektom opisuje se promena koja je nastala u skladištu prilikom emitovanja `storage` događaja. `storage` događaj se emituje kada se stanje skladišta promeni – bilo da je reč o dodavanju nove vrednosti ili izmeni ili brisanju postojeće.

`storage` događaj je moguće slušati definisanjem funkcije kao vrednosti `onstorage` svojstva:

```
window.onstorage = function (event) {
      // handle storage event
}
```

Naravno, na događaj je moguće izvršiti pretplatu i korišćenjem `addEventListener()` metode:

```
window.addEventListener("storage", function (event) {
      // handle storage event
});
```

Parametar koji funkcija za obradu događaja dobija (`event` promenljiva u primeru) automatski se od strane pregledača popunjava referencom na `StorageEvent` objekat sa informacijama o promeni do koje je došlo u skladištu. `StorageEvent` poseduje sledeća svojstva:

- `key` – ključ čija se vrednost promenila ili `null` ukoliko je pozvana metoda `clear()`;
- `oldValue` – stara vrednosti ili `null` ukoliko je vrednost upravo dodata;
- `newValue` – nova vrednost ili `null` ukoliko je vrednost upravo izbrisana;
- `url` – URL dokumenta u kome se dogodila promena nad skladištem;
- `storageArea` – referenca na skladište u kome je došlo do promene.

Na osnovu osobina `StorageEvent` objekta, sada možemo i da napišemo logiku funkcije za obradu `storage` događaja:

```
    window.onstorage = function (event) {

        if (event.key === null) {
            console.log("Whole storage is cleared!");
            return;
        }

        if (event.oldValue === null) {
            console.log("New value added to storage: " + event.key + " " +
event.newValue);
            return;
        }

        if (event.newValue === null) {
            console.log("Value removed from stroage: " + event.key + " " +
event.oldValue);
            return;
        }

        console.log("Value with key: " + event.key + " updated. Old value: "
+ event.oldValue + " New value: " + event.newValue);
    }
```

Logikom unutar upravo prikazane funkcije za obradu `storage` događaja obrađeni su svi scenariji ažuriranja podataka unutar skladišta:

- kada svojstvo `key` ima vrednosti `null`, to znači da je kompletno skladište očišćeno metodom `clear()`, pa se aktivira prvi if uslovni blok;
- kada svojstvo `oldValue` ima vrednost `null`, izvršeno je dodavanje nove vrednosti, pa se aktivira drugi if uslovni blok;
- kada je svojstvo `newValue` `null`, izvršeno je brisanje jedne vrednosti, pa se aktivira treći if uslovni blok;

- na kraju, kada nijedno od svojstava `key`, `oldValue` i `newValue` nema vrednost `null`, znači da je obavljeno ažuriranje vrednosti postojećeg ključa, pa se izvršava poslednja naredba koja je izvan uslovnih blokova.

Pre nego što primer isprobamo unutar web pregledača, neophodno je razumeti najznačajniju osobinu `storage` događaja, o kojoj do sada još nije bilo reči:

`storage` događaj emituje se samo ukoliko je do promene unutar skladišta došlo unutar nekog drugog dokumenta.

Zbog ove značajne osobine `storage` događaja, naredni primer je neophodno testirati korišćenjem dva prozora pregledača.

Primer – Praćenje promene unutar skladišta korišćenjem storage događaja

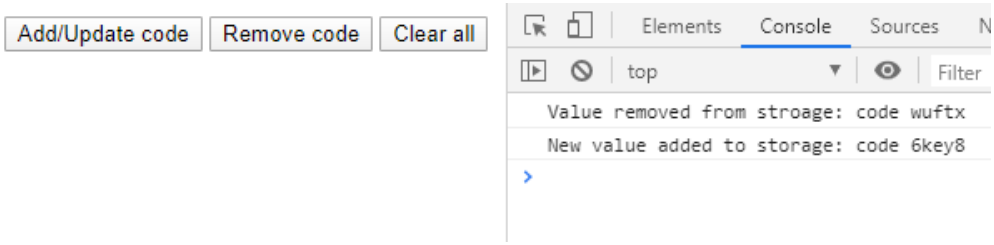
Testiranje `storage` događaja biće ilustrovano na jednom primeru, u kome će korisnik imati mogućnost da inicira operacije dodavanja, ažuriranja i brisanja zapisa unutar local storage skladišta. Akcije će se pokretati klikom na odgovarajuće dugme na stranici i na taj način će biti rukovano local storage zapisom sa ključem `code`.

Evo kako izgleda kod kompletne HTML stranice:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Local Storage Demo</title>
    </head>
    <body>

        <button id="add-update">Add/Update code</button>
        <button id="remove">Remove code</button>
        <button id="clear">Clear all</button>

        <script>
            var addUpdateButton = document.getElementById("add-update");
            var removeButton = document.getElementById("remove");
            var clearButton = document.getElementById("clear");

            addUpdateButton.onclick = function () {
                localStorage.setItem('code',
Math.random().toString(36).substring(7));
            }

            removeButton.onclick = function () {
                localStorage.removeItem('code');
            }

            clearButton.onclick = function () {
                localStorage.clear();
            }
```

```
            window.onstorage = function (event) {
                if (event.key === null) {
                    console.log("Whole storage is cleared!");
                    return;
                }

                if (event.oldValue === null) {
                    console.log("New value added to storage: " + event.key +
" " + event.newValue);
                    return;
                }

                if (event.newValue === null) {
                    console.log("Value removed from stroage: " + event.key +
" " + event.oldValue);
                    return;
                }

                console.log("Value with key: " + event.key + " updated. Old
value: " + event.oldValue + " New value: " + event.newValue);

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

Primer će unutar web pregledač stvoriti efekat kao na slici 12.3.



Slika 12.3. Izgled primera za testiranje storage događaja unutar pregledača

Kao što je već rečeno, neophodno je da prikazani HTML dokument otvorite u dva zasebna taba. Kada unutar jednog taba obavite intervenciju nad skladištem, `storage` događaj neće biti emitovan u tom tabu, već u svim ostalim tabovima u kojima je otvoren prikazani HTML dokument.

Klikom na dugme *Add/Update code* obavlja se kreiranje zapisa sa ključem `code` ukoliko takav zapis ne postoji, ili njegovo ažuriranje ukoliko postoji. Svakim novim klikom na dugme *Add/Update code* obavlja se generisanje novog koda.

Brisanje zapisa sa `code` ključem se obavlja klikom na dugme *Remove code*. Na taj način se iz skladišta uklanja samo `code` zapis, dok svi ostali parovi ključeva i vrednosti koji eventualno postoje unutar skladišta ostaju nepromenjeni.

Na kraju, klikom na dugme *Clear all* obavlja se brisanje svih zapisa iz skladišta.

Upravo prikazani primer nije moguće realizovati korišćenjem session storagea. Naime, već je rečeno da svaki dokument poseduje sopstvenu instancu session storagea (odnosno `Storage` objekta koji predstavlja skladište). Stoga otvaranje jednog HTML dokumenta u dva različita taba, ima za efekat stvaranje dva potpuno nezavisna session skladišta, koja međusobno ne mogu da komuniciraju. Ukoliko se pitate da li je onda uopšte moguće iskoristiti `storage` događaj za rukovanje session storageom, odgovor je – da, ali samo kada se unutar jednog dokumenta (taba), nalazi više `iframe` elemenata unutar kojih su učitani dokumenti sa istim originom. Stoga, ukoliko prikazani primer želite da isprobate u kombinaciji sa session skladištem, napravite još jedan HTML dokument, unutar koga ćete korišćenjem dva `iframe` elementa uključiti već prikazan HTML dokument. Naravno, neophodno je da izmenite `localStorage` svojstvo u `sessionStorage` i bićete u mogućnosti da dobijete `storage` događaje do kojih dolazi prilikom promena na session storageu.

Rezime

- Kolačići su dugo bili jedini način da se određeni podaci sačuvaju na klijentu.
- Mehanizmi za čuvanje podataka na klijentu koji su direktna alternativa kolačićima nazivaju se local storage i session storage.
- Web Storage API je objedinjeni naziv za skup funkcionalnosti koje web pregledači obezbeđuju kako bi se rukovalo local i session skladištima.
- Podaci local i session skladišta čuvaju se u formi parova ključeva i vrednosti.
- Podaci iz local i session skladišta nikada se automatski ne prosleđuju serveru.
- Session storage kreira se za svaku instancu jednog origina i njegovi podaci se čuvaju sve dok je aktivna sesija web pregledača.
- Jedina akcija koju session storage podaci mogu da prežive jeste osvežavanje stranice.
- Local storage deli se između svih aktivnih tabova sa stranicama istog origina.
- Jedini način za brisanje local storage podataka jeste da oni budu obrisani od strane JavaScript jezika ili eksplicitno od strane web pregledača, brisanjem keša.
- Svojstva `Window` objekta `Window.localStorage` i `Window.sessionStorage` omogućavaju pristup konkretnim skladištima.
- Podaci local i session skladišta mogu se upisati ili ažurirati korišćenjem metode `setItem()`.
- Local storage i session storage omogućavaju čuvanje podataka u `string` obliku.
- Podaci iz local i session skladišta mogu se čitati korišćenjem metode `getItem()`.
- Brisanje zapisa iz local i session skladišta moguće je obaviti korišćenjem metode `removeItem()`.
- `StorageEvent` objektom opisuje se promena koja je nastala u skladištu prilikom emitovanja `storage` događaja.
- `storage` događaj emituje se samo ukoliko je do promene unutar skladišta došlo unutar nekog drugog dokumenta.