IndexedDB

Web Storage API, odnosno local storage i session storage skladišta ilustrovana u prethodnoj lekciji, predstavlja veoma moćnu, modernu alternativu kolačićima za čuvanje manje količine podataka unutar web pregledača. Ipak, u situacijama koje iziskuju čuvanje veće količine strukturiranih podataka na klijentu, Web Storage API može biti prilično ograničavajuć. Prvo ograničenje se tiče količine podataka koju je moguće sačuvati. Različiti pregledači poseduju različita ograničenja, ali se može reći da se u takvim skladištima najčešće ne može čuvati više od 10MB podataka, kumulativno. Dalje, u prethodnoj lekciji ste mogli da vidite da Web Storage API poseduje i ograničenja po pitanju formata u kome se podaci čuvaju (i ključevi i vrednosti moraju biti `string` tipa). Zbog takvih ograničenja, moderni web pregledači poseduju još jedan mehanizam za čuvanje podataka na klijentu. Reč je o IndexedDB bazi podataka.

Šta je baza podataka?

Baza podataka je kolekcija strukturiranih informacija. Osnovna uloga baze je da upravlja velikom količinom podataka i da na organizovan i strukturiran način omogući korisniku rukovanje podacima (slika 13.1).

Slika 13.1. Baza podataka objedinjuje različite napredne funkcionalnosti rada sa podacima

Baze podataka poseduju različite mehanizme koji omogućavaju da se velika količina podataka logički organizuje, odnosno strukturira i da se takvi podaci na lak način pretražuju, filtriraju i sortiraju. Takođe, baze podataka poseduju jednostavne mehanizme za obavljanje osnovnih operacija nad podacima, što podrazumeva kreiranje, brisanje, ažuriranje i čitanje. Na kraju, baze podataka karakterišu se i naprednim mehanizmima koji omogućavaju da se podacima unutar njih rukuje na što optimalniji način, čime se značajno unapređuju performanse.

Baze podataka su pojam koji se najčešće vezuje za backend programiranje. Jednostavno, backend logika web aplikacija gotovo da ne može da se zamisli bez baza podataka, koje se koriste za pouzdano i dugotrajno čuvanje podataka kojima web aplikacije rukuju. Svakako najpopularniji sistem za upravljanje bazama podataka je MySQL. Reč je o sistemu za upravljanje relacionim bazama podataka, koji se zasniva na upotrebi jednog posebnog upitnog jezika – SQL. Ipak, pored relacionih baza podataka koje koriste SQL, danas veliku popularnost imaju i različiti sistemi koji se zasnivaju na drugačijim modelima, pre svih na modelima koji podatke čuvaju u različitim formama parova ključeva i vrednosti. Najpopularniji takvi sistemi su MongoDB, Apache CouchDB, ArangoDB...

Šta je IndexedDB?

IndexedDB je osnovna klijentska baza podataka web aplikacija. To praktično znači da je IndexedDB baza podataka koja postoji unutar web pregledača i na raspolaganju je web aplikacijama, koje je mogu koristiti za čuvanje podataka na klijentu.

Za razliku od najpopularnijih sistema za upravljanje bazama podataka koji se koriste na backendu, IndexedDB je objektno orijentisana baza podataka, u kojoj se podaci čuvaju u formi parova ključeva i vrednosti, baš kao što je to bio slučaj i kod Web Storage API-a. Takođe, IndexedDB ne koristi SQL, već obezbeđuje sopstvene mehanizme za rukovanje podacima koji su bliski objektno orijentisanom modelu JavaScripta.

Iako se IndexedDB, baš kao i Web Storage API, zasniva na čuvanju podataka u formi parova ključeva i vrednosti, on poseduje brojne prednosti koje olakšavaju rad sa velikom količinom podataka:

- ključevi i vrednosti nisu ograničeni na `string` tip; vrednosti mogu biti praktično bilo kog tipa, dok je za definisanje ključeva na raspolaganju nekoliko tipova podataka;
- IndexedDB obezbeđuje rukovanje podacima kroz transakcije, što dodatno osigurava pouzdanost prilikom rada sa podacima;
- IndexedDB obezbeđuje različite funkcionalnosti za pretragu podataka u bazi;
- IndexedDB podržava indekse;
- IndexedDB može da sačuva mnogo veću količinu podataka od local storage i session storage skladišta; većina web pregledača ne definiše limit za ukupnu veličinu IndexedDB baze.

Zbog svojih osobina, IndexedDB se najčešće koristi za takozvane *offline web aplikacije*, odnosno aplikacije koje radi u nepovezanom okruženju, delimično ili u potpunosti. Naime, određene web aplikacije mogu samo jednom da dobave veliku količinu podataka, a zatim da takvim podacima rukuju lokalno uz podršku IndexedDB baze podataka. U takvim situacijama, web aplikacija može samo povremeno da sinhronizuje lokalne podatke sa podacima na udaljenom serveru.

Korišćenje IndexedDB-a podrazumeva sprovođenje sledećih koraka:

- otvaranje baze podataka;
- kreiranje strukture baze, definisanjem objektnih skladišta;
- kreiranje transakcije unutar koje će biti obavljene operacije nad podacima, kao što su upisivanje, čitanje, izmena i brisanje;
- čekanje na završetak operacije, slušanjem određenih događaja;
- obrada dobijenih podataka.

Otvaranje baze podataka

Korišćenje IndexedDB baze podataka započinje otvaranjem konekcije. Konekcija se otvara korišćenjem metode `open()`.

Metoda open()

Otvaranje konekcije se obavlja korišćenjem metode `open()` objekta `IDBFactory`. Ovom objektu se pristupa korišćenjem `window.indexedDB` svojstva:

```
var request = window.indexedDB.open(name, version);
```

Metoda `open()` prihvata dva parametra:

- `name` – naziv baze podataka, u `string` obliku
- `version` – verzija baze podataka, koja se definiše pozitivnom celobrojnom vrednošću; podrazumevana vrednost je 1

Primer naredbe za otvaranje baze podataka može da izgleda ovako:

```
var request = window.indexedDB.open("MyDatabase", 1);
```

Pozivanje metode `open()` ne otvara konekciju sa bazom podataka istog trenutka, već kao svoju povratnu vrednost isporučuje objekat tipa `IDBOpenDBRequest`. Reč je o objektu koji omogućava pretplatu na nekoliko događaja koji su veoma značajni za početak korišćenja IndexedDB baze podataka:

```
var request = window.indexedDB.open("MyDatabase", 1);

request.onupgradeneeded = function (event) {
        // triggers if the client had no database
        // ...perform initialization...
};

request.onsuccess = function (event) {
        // do something with request.result
};

request.onerror = function (event) {
        // do something with request.errorCode
};
```

Možete videti da prikazani kod ilustruje i pretplatu na nekoliko događaja. Reč je o događajima koji se aktiviraju u različitim situacijama prilikom otvaranja baze podataka.

Događaji za obradu zahteva otvaranja baze podataka

Prilikom otvaranja baze podataka, može doći do emitovanja sledećih događaja:

- `upgradeneeded` – događaj koji se aktivira ukoliko se baza podataka otvara prvi put ili ukoliko se vrši njeno ažuriranje; nešto kasnije će biti više reči o pojmu verzije baze podataka
- `success` – aktivira se kada je baza podataka spremna za korišćenje
- `error` – emituje se kada dođe do greške prilikom otvaranja baze podataka; kod greške se može dobiti `request.errorCode` svojstvom

Sada ćemo definisati i osnovnu logiku funkcija za obradu tri upravo definisana događaja:

```
var db;
var request = window.indexedDB.open("MyDatabase", 1);

request.onupgradeneeded = function (event) {
    // triggers if the client had no database
    // ...perform initialization...
};

request.onsuccess = function (event) {
    db = event.target.result;
};

request.onerror = function (event) {
    console.error("Error", event.target.error);
};
```

Sada je definisana logika `onsuccess` i `onerror` funkcija. S obzirom na to da se funkcija `onsuccess` aktivira kada je baza podataka spremna za korišćenje, unutar `onsuccess` funkcije dolazi se do objekta baze podataka. Reč je o objektu tipa `IDBDatabase` čiju smo referencu u primeru smestili unutar promenjive `db`.

U prikazanom kodu definisana je i jednostavna logika `onerror` funkcije. Ona se aktivira ukoliko dođe do greške prilikom otvaranja baze podataka. Do greške prilikom otvaranja baze podataka najčešće dolazi ukoliko pregledač ili korisnik ne dozvoli web sajtu da kreira lokalnu IndexedDB bazu podataka (na primer, prilikom surfovanja u inkognito modu, Firefox pregledač uopšte ne dozvoljava korišćenje IndexedDB baza podataka). Do poruke greške se dolazi korišćenjem svojstva `error`, `request` objekta.

U prikazanom primeru možete da vidite da logika `onupgradeneeded` metode još uvek nije definisana. Kako bismo i nju definisali, neophodno je da se prethodno upoznamo sa još nekim značajnim pojmovima.

Šema i verzije baze podataka

Do sada je nekoliko puta spomenut pojam verzije baze podataka. Mogli ste da vidite da se verzija isporučuje i metodi `open()` prilikom otvaranja konekcije, a da promene verzije aktiviraju emitovanje događaja `upgradeneeded`.

Pojam verzije baze podataka usko je povezan sa još jednim pojmom, sa kojim se do sada nismo susretali. Reč je o pojmu šeme baze podataka (*database schema*). **Šema baze** podataka predstavlja njenu osnovnu strukturu. Naime, prilikom rada sa bazama podataka, pre nego što se pređe na upisivanje podataka, neophodno je kreirati njenu strukturu. Struktura relacionih baza podataka najčešće se definiše korišćenjem tabela. Kada je posredi IndexedDB, struktura se definiše na nešto drugačiji način – definisanjem objektnih skladišta.

U nekom trenutku se može javiti potreba za promenom prethodno kreirane strukture i upravo u takvim situacijama verzija baze podataka stupa na scenu. Tako je verzija baze podataka, koja se definiše kao drugi parametar `open()` metode, zapravo verzija njene šeme.

Kreiranje strukture baze podataka – objektna skladišta

Kao što je rečeno u prethodnim redovima, kako bi se unutar IndexedDB baze podataka mogli upisivati podaci, prethodno je neophodno kreirati njenu šemu. Šema je zapravo struktura baze podataka, a kod IndexedDB baza podataka ona se definiše kreiranjem objektnih skladišta.

IndexedDB baze podataka mogu imati proizvoljni broj objektnih skladišta, odnosno za svaki tip objekata po jedno takvo skladište. Stoga, ukoliko je potrebno da naša aplikacija na klijentu čuva podatke o studentima, profesorima, predavanjima i predmetima, kreiraćemo četiri objektna skladišta.

Svako objektno skladište može da prihvati proizvoljni broj parova ključeva i vrednosti. Tako osnovna struktura jedne IndexedDB baze podataka može da izgleda kao na slici 13.2.

Slika 13.2. Osnovna struktura IndexedDB baze podataka

Slika 13.2. ilustruje jednu IndexedDB bazu podataka sa tri objektna skladišta sa različitom količinom podataka.

Definisanje šeme, odnosno strukture baze podataka jeste operacija koja se ne obavlja prilikom svakog otvaranja jedne web aplikacije. Naime, šema se definiše jednom, a zatim se objektna skladišta dobijena na taj način koriste za upisivanje, čitanje, izmenu i brisanje podataka. Stoga je kod kojim se kreira šema potrebno izvršiti samo jednom, a ne prilikom svakog pokretanja jedne aplikacije. Upravo zbog toga postoji nešto ranije prikazana metoda `onupgradeneeded` – koja se aktivira ukoliko se baza podataka otvara prvi put ili ukoliko je došlo do promene verzije šeme. Štaviše, `onupgradeneeded` funkcija jedino je mesto na kome je moguće napisati kod za kreiranje šeme IndexedDB baze podataka.

Kao što je prikazano na slici 13.2, šema jedne IndexedDB baze podataka realizuje se kreiranjem objektnih skladišta. Objektno skladište je moguće definisati korišćenjem metode `createObjectStore()` nad prethodno dobijenim objektom tipa `IDBDatabase`.

Metoda za kreiranje objektnih skladišta - createObjectStore()

Metoda `createObjectStore()` ima sledeću sintaksu:

```
createObjectStore(name[, keyOptions]);
```

Metoda `createObjectStore()` može da prihvati dva parametra, pri čemu je samo prvi parametar obavezan:

- `name` – naziv objektnog skladišta
- `keyOptions` – objekat kojim je moguće definisati dodatne osobine skladišta, korišćenjem jednog od sledeća dva svojstva:
    - `keyPath` – naziv objektnog svojstva koje će biti korišćeno kao ključ, odnosno identifikaciona vrednost koja će biti korišćena za pristup objektima; vrednosti definisanog svojstva moraju biti jedinstvene za sve objekte unutar skladišta
    - `autoIncrement` – svojstvo `boolean` tipa, kojim je moguće uključiti automatsko generisanje ključeva za objekte koji se smeštaju unutar objektnog skladišta; vrednost koja se automatski generiše takođe se automatski uvećava za jedan za svaki naredni objekat koji se dodaje u skladište; zbog toga se svojstvo i zove `autoIncrement`

Parametar `keyOptions` je moguće izostaviti i u takvom slučaju objekti koji se budu dodavali unutar objektnog skladišta neće podrazumevano posedovati mehanizam za kreiranje ključeva, već je ključeve neophodno definisati za svaki objekat pojedinačno, prilikom njihovog upisivanja u bazu. O tome će biti reči u nastavku lekcije, kada dođemo do pristupa za upisivanje podataka u bazu.

Bitno je takođe reći da je `keyOptions` parametrom moguće definisati samo jedan od dva prikazana načina za definisanje ključeva, odnosno ili `keyPath` ili `autoIncrement`.

Kod za kreiranje našeg prvog objektnog skladišta može da izgleda ovako:

```
var db;
var request = window.indexedDB.open("MyDatabase", 1);

request.onupgradeneeded = function (event) {

      db = event.target.result;

      var objectStore = db.createObjectStore("students", { keyPath:
"usi" });

};
```

Isečak koda prikazuje logiku za kreiranje objektnog skladišta. Kreiranje se obavlja unutar `onupgradeneeded` funkcije. Unutar nje se prvo postavlja vrednost `db` svojstva. Iako je identična logika nešto ranije obavljena unutar `onsuccess` funkcije, bitno je znati da će se prilikom prvog pokretanja klijentske logike, dok IndexedDB baza podataka još nije kreirana, prvo aktivirati `onupgradeneeded` funkcija, pa tek nakon toga i funkcija `onsuccess`.

Zatim se unutar `onupgradeneeded` funkcije obavlja kreiranje objektnog skladišta sa nazivom `students`. Prilikom kreiranja objektnog skladišta definiše se i naziv objektnog svojstva koje će biti korišćeno kao ključ objekata (ključ je svojstvo `usi`, što je skraćenica za *unique student identifier*, odnosno jedinstveni identifikator studenta – drugim rečima, broj indeksa).

Ukoliko ne želimo da prilikom definisanja objektnog skladišta za definisanje ključa iskoristimo neko od svojstava unutar objekta, generisanje ključeva možemo da prepustimo web pregledaču, na sledeći način:

```
var objectStore = db.createObjectStore("students", { autoIncrement: true
});
```

Na ovaj način će IndexedDB API samostalno definisati ključeve svakog objekta koji se upiše unutar skladišta (prvi upisani objekat će imati ključ 1, drugi ključ 2, treći 3 itd.).

Pitanje

Ukoliko je došlo do promene verzije baze podataka, aktivira se funkcija:

- **onupgradeneeded**
- onupgraderequired
- ondowngrade
- ondowngradeneeded

Objašnjenje:

Događaj upgradeneeded i pripadajuća funkcija onupgradeneeded aktiviraju se ukoliko se baza podataka otvara prvi put ili ukoliko se vrši njeno ažuriranje.

Upisivanje podataka

Svaka operacija nad podacima IndexedDB baze (upisivanje, čitanje, ažuriranje i brisanje) započinje kreiranjem transakcije.

IndexedDB transakcije

Obavljanje bilo koje operacije nad IndexedDB podacima zahteva postojanje transakcije. Transakcija se kreira korišćenjem metode **transaction()**:

```
db.transaction(store[, type]);
```

Prilikom kreiranja transakcije, potrebno je navesti naziv objektnog skladišta kojim će se operisati, a opciono i tip transakcije:

- `store` – nazivi jednog ili više objektnih skladišta kojima će moći da se rukuje unutar transakcije; ukoliko je potrebno da transakcija rukuje jednim skladištem, vrednost je moguće definisati kao `string`; veći broj objektnih skladišta se može definisati kao niz `string` vrednosti
- `type` – tip transakcije; IndexedDB poznaje nekoliko tipova transakcija:
    - `readonly` – transakcija koja se može koristiti samo za čitanje podataka; ovo je podrazumevani tip transakcije, odnosno, ukoliko se ne navede tip, dobija se *readonly* transakcija
    - `readwrite` – transakcija koja se može koristiti za čitanje ili promenu podataka
    - `versionchange` – transakcija koja se koristi za kreiranje šeme baze podataka; reč je o tipu transakcije koji mi samostalno ne možemo da kreiramo, već IndexedDB API kreira transakciju automatski, prilikom otvaranja baze podataka, odnosno prilikom pozivanja metode `open()`; ovakav tip transakcije automatski je na raspolaganju `onversionchange` funkciji, odnosno, kompletan kod koji se piše unutar takve funkcije izvršava se unutar transakcije ovog tipa; to je još jedan razlog zašto se kod za definisanje šeme baze podataka mora navesti isključivo unutar `onversionchange` funkcije

Metoda `transaction()` kao svoju povratnu vrednost emituje objekat tipa `IDBTransaction`. Ovaj objekat je dalje moguće koristiti za dolazak do objektnog skladišta i praćenje događaja transakcije.

Naredba za kreiranje transakcije, unutar koje ćemo obaviti dodavanje prvih podataka u IndexedDB, može da izgleda ovako:

```
let transaction = db.transaction("students", "readwrite");
```

Na ovaj način je kreirana transakcija unutar koje će moći da se rukuje `students` skladištem, a kako bismo mogli da obavimo upisivanje podataka, tip transakcije je postavljen ne `readwrite`.

Nakon kreiranja transakcije potrebno je doći do reference na objektno skladište kojim želimo da operišemo. Unutar transakcije je moguće raditi isključivo sa skladištima koja su navedena kao prvi parametar prilikom kreiranja transakcije. U našem primeru postoji samo jedno objektno skladište (`students`), a dolazak do reference na njega izgleda ovako:

```
var transaction = db.transaction("students", "readwrite");
var studentsObjectStore = transaction.objectStore("students");
```

Nakon dobijanja reference na objektno skladište, moguće je obaviti prvo upisivanje podataka u IndexedDB bazu.

Metoda add()

Za dodavanje podataka u IndexedDB bazu koristiti se metoda **add()**:

```
add(value, [key])
```

Metoda `add()` može da prihvati dva parametra, od kojih je samo prvi obavezan:

- `value` – vrednost (prosta vrednost ili objekat) koja se upisuje u bazu
- `key` – neobavezni parametar kojim je moguće definisati ključ objekta, ukoliko to nije učinjeno prilikom definisanja skladišta (korišćenjem `keyPath` ili `autoIncrement` svojstava)

Metoda `add()` dodaje vrednost unutar skladišta, a ukoliko unutar skladišta već postoji vrednost sa definisanim ključem, dolazi do podizanja izuzetka tipa `ConstraintError`.

Metoda `add()`, ali i sve ostale metode za manipulaciju podacima sa kojima ćemo se upoznati u nastavku ove lekcije, kao svoju povratnu vrednost emituju objekat zahteva, koji se može koristiti za pretplatu na događaje uspešnog i neuspešnog obavljanja operacija nad podacima:

- `success` – događaj koji se emituje kada se operacija uspešno izvrši
- `error` – događaj koji se emituje kada dođe do greške prilikom operacije nad podacima

Objekat koji se dobija kao povratna vrednost metoda za manipulaciju podacima poseduje i nekoliko značajnih svojstava:

- `request.result` – rezultat operacije nad podacima; u slučaju dodavanja to će biti vrednost koja će predstavljati ključ objekta
- `request.error` – greška do koje je došlo prilikom obavljanja operacije nad podacima

Kod za upisivanje prvog objekta unutar IndexedDB baze može da izgleda ovako:

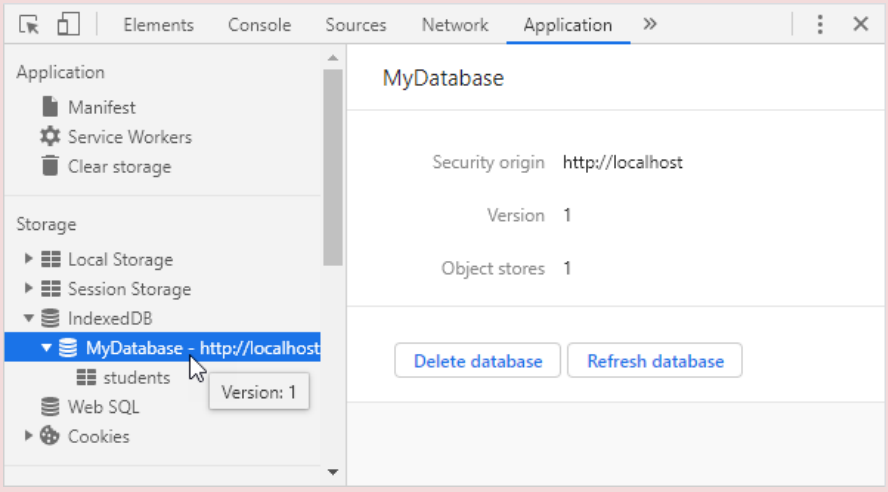
```
var transaction = db.transaction("students", "readwrite");

var studentsObjectStore = transaction.objectStore("students");

let student = {
   usi: "v43v2f",
   name: "John Steel",
   email: "email@mysite.com",
   age: 22
}

let request = studentsObjectStore.add(student);

request.onsuccess = function () {
   console.log("Student added to the store", request.result);
};

request.onerror = function () {
   console.log("Error", request.error);
};
```

Prikazanim kodom obavlja se upisivanje prvog objekta unutar IndexedDB baze podataka. U bazu se upisuje objekat koji predstavlja jednog studenta, sa svojstvima `usi`, `name`, `email` i `age`.

Nakon pozivanja metode `add()`, obavljena je pretplata na `success` i `error` događaje. Odgovarajuća funkcija će biti aktivirana u zavisnosti od uspešnosti operacije dodavanja podataka.

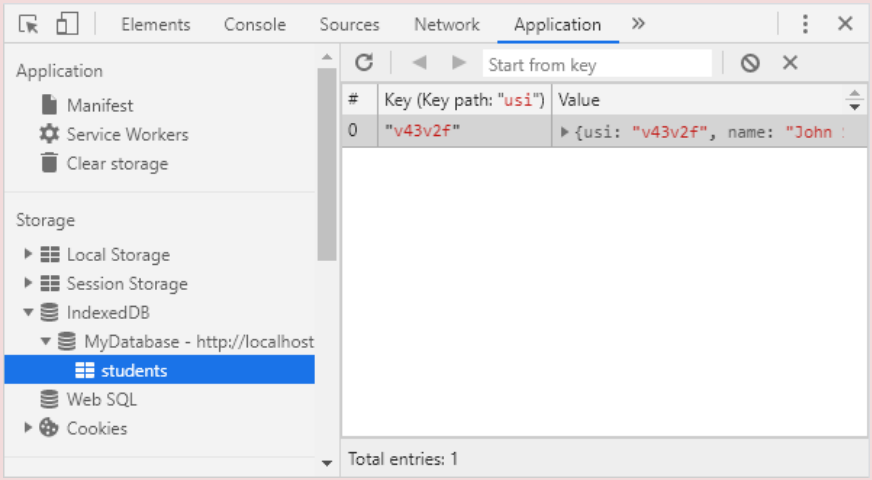
Pregled podataka IndexedDB baze korišćenjem Chrome pregledača

Strukturu i podatke IndexedDB baze moguće je veoma lako pregledati korišćenjem browsera Chrome. Dovoljno je unutar Developer Tools panela odabrati tab *Application*, a zatim iz menija sa leve strane odgovarajuću IndexedDB bazu podataka (slika 13.3).



Slika 13.3. Pregled IndexedDB baze unutar Chrome pregledača

Sa slike 13.3. možemo da vidimo da je tekućem dokumentu na raspolaganju jedna baza podataka, sa nazivom *MyDatabase*, i da takva baza podataka poseduje jedno objektno skladište – *students*. Klikom na objektno skladište, unutar desne polovine panela se prikazuju podaci objektnog skladišta (slika 13.4).



Slika 13.4. Pregled IndexedDB baze unutar Chrome pregledača (2)

Na slici 13.4. možete videti podatak koji smo upravo upisali unutar IndexedDB baze podataka.

Brisanje podataka

Brisanje prethodno upisanih podataka može se obaviti korišćenjem metode `delete()`.

Metoda delete()

Sintaksa metode za brisanje zapisa iz IndexedDB baze izgleda ovako:

```
delete(query)
```

Metoda `delete()` prihvata jedan parametar, koji se odnosi na uslov koji objekat iz baze treba da zadovolji kako bi bio obrisan. Najčešće se kao vrednost parametra prosleđuje ključ objekta.

Baš kao i metode za unos zapisa, ova metoda kao svoju povratnu vrednost emituje objekat koji je moguće koristiti za dobijanje dojava o uspešnosti operacije brisanja. Koriste se sledeći događaji i svojstva:

- `success` – događaj koji se emituje kada se operacija uspešno izvrši
- `error` – događaj koji se emituje kada dođe do greške prilikom operacije nad podacima
- `request.result` – rezultat operacije nad podacima; u slučaju dodavanja to će biti vrednost koja će predstavljati ključ objekta
- `request.error` – greška do koje je došlo prilikom obavljanja operacije nad podacima

Kod za brisanje objekta koji je u prethodnim redovima upisan u bazu izgleda ovako:

```
var request = db.transaction(["students"], "readwrite")
.objectStore("students")
.delete("v43v2f");

request.onsuccess = function (event) {
console.log("Student successfully deleted from DB.");
};

request.onerror = function () {
console.log("Error", request.error);
};
```

Sada smo za obavljanje pripremnih radnji koje prethode svakoj operaciji nad podacima (otvaranje transakcije i dolazak do instance objektnog skladišta) iskoristili nešto drugačiji pristup, kako bismo ilustrovali različite načine za obavljanje takvog posla. Prva razlika je u načinu na koji se definiše prvi parametar metode za otvaranje transakcije. Umesto `string` vrednosti sada je definisan niz sa jednom `string` vrednošću. Rezultat je isti kao i kada je `string` vrednost koja predstavlja naziv skladišta bila navedena izvan niza.

Druga razlika odnosi se na nadovezivanje metoda `transaction()`, `objectStore()` i `delete()`. U prethodnom primeru smo svaki od poziva smeštali unutar zasebne naredbe kako biste na najbolji način razumeli njihove osobine. Ipak, u realnom radu se najčešće pribegava nadovezivanju ovakvih metoda, kako bi se dobio kompaktniji i razumljiviji kod.

Za obavljanje brisanja iskorišćena je metoda `delete()` i njoj je kao parametar prosleđen ključ objekta koji želimo da obrišemo. S obzirom na to da smo ranije definisali da će ključevi biti vrednosti `usi` svojstava, sada je vrednost takvog svojstva definisana kao uslov za brisanje. Drugim rečima, objekat čije svojstvo `usi` ima prosleđenu vrednost biće obrisan iz baze podataka.

Pored naredbe za brisanje, definisane su i dve funkcije koje će se aktivirati kada se brisanje završi uspešno, odnosno neuspešno, respektivno.

Brisanje svih podataka jednog skladišta

IndexedDB API omogućava brisanje svih zapisa iz nekog skladišta odjednom i to korišćenjem metode **clear()**:

```
var request = db.transaction(["students"], "readwrite")
.objectStore("students")
.clear();
```

Čitanje jednog zapisa

Čitanje jednog zapisa iz IndexedDB baze se može obaviti korišćenjem metode `get()`.

Metoda get()

Metoda `get()` se koristi za čitanje jednog zapisa iz objektnog skladišta IndexedDB baze. Sintaksa metode `get()` izgleda ovako:

```
get(query)
```

Kao i kod metode za brisanje, metoda `get()` prihvata jedan parametar, kojim se definiše kriterijum za čitanje. Najčešće se prosleđuje ključ objekta koji je potrebno pročitati.

Ukoliko ste u prethodnom primeru obavili brisanje jedinog zapisa koji se nalazio unutar objektnog skladišta, pre isprobavanja koda koji će biti prikazan ponovo obavite upisivanje nekog objekta u bazu.

Kod za čitanje upisanog objekta studenta može da izgleda ovako:

```
db.transaction("students").objectStore("students").get("v43v2f").onsucce
ss = function (event) {
let studentObj = event.target.result;
console.log(studentObj.usi);
console.log(studentObj.name);
console.log(studentObj.email);
console.log(studentObj.age);
};
```

Prikazana naredba predstavlja najkompaktniji oblik koji je moguće koristiti za kreiranje transakcije, dolazak do skladišta i upućivanje komande za vršenje manipulacije podacima. Pozivi svih metoda su nadovezani, a prikazani pristup nam omogućava da dobijemo dojavu samo o uspešnom završetku čitanja vrednosti.

Iz prikazane naredbe možete da vidite da se prilikom čitanja podataka iz objektnog skladišta oni isporučuju unutar `onsuccess` metode. Do pročitanog objekta dolazimo korišćenjem svojstva `event.target.result`. Tako na kraju primer unutar konzole proizvodi sledeći ispis:

```
v43v2f
John Steel
email@mysite.com
22
```

U primeru je bitno da primetite još jedan mali detalj – transakcija je sada `readonly`, zato što njen tip nije eksplicitno definisan i zato što se obavlja čitanje podataka. Drugim rečima, nema potrebe za `readwrite` transakcijom.

Ažuriranje podataka

Upravo pročitani podatak moguće je ažurirati, a zatim tako ažuriran podatak vratiti u bazu podataka. Proces ažuriranja će biti ilustrovan u narednim redovima. Metoda koja se koristi za ažuriranje podataka je `put()`.

Metoda put()

Sintaksa metode `put()` izgleda ovako:

```
put(value, [key])
```

Metoda `put()` može da prihvati dva parametra, od kojih je samo prvi obavezan:

- `value` – vrednost (prosta vrednost ili objekat) koja se upisuje u bazu
- `key` – neobavezni parametar kojim je moguće definisati ključ objekta, ukoliko to nije učinjeno prilikom definisanja skladišta (korišćenjem svojstava `keyPath` ili `autoIncrement`)

Metoda `put()` se može koristiti za dodavanje ili ažuriranje zapisa u bazi. Kada vrednost sa definisanim ključem ne postoji u bazi, ona se dodaje, a ukoliko unutar skladišta već postoji vrednost sa definisanim ključem, vrednost se upisuje preko postojeće vrednosti.

Kod za ažuriranje prethodno upisanog objekta može da izgleda ovako:

```
var studentsObjectStore = db.transaction("students",
"readwrite").objectStore("students");

studentsObjectStore.get("v43v2f").onsuccess = function (event) {
     let studentObj = event.target.result;

     studentObj.age = 25;

     var requestUpdate = studentsObjectStore.put(studentObj);

requestUpdate.onerror = function (event) {
     console.log("Error", requestUpdate.error);
};
requestUpdate.onsuccess = function (event) {
      console.log("Student successfully updated.");
};

};
```

Primer započinje kreiranjem jedne transakcije. Transakcija je ovoga puta `readwrite`, zato što će biti korišćena i za čitanje, ali i za upisivanje podataka u bazu. Naime, prvo se obavlja čitanje objekta studenta koji je upisan u bazu, a zatim njegovo ažuriranje i ponovno upisivanje.

Logika za ažuriranje je smeštena unutar `onsuccess` funkcije, koja se aktivira kada se objekat pročita iz baze. Zatim se obavlja ažuriranje jednog svojstva takvog objekta (svojstva `age`). Nakon toga se obavlja pozivanje metode `put()`, koja će ažurirati postojeći zapis u bazi.

Kao i kod ostalih metoda za manipulaciju podacima, pozivanje metode `put()` proizvodi objekat zahteva koji je u primeru iskorišćen za pretplatu na `success` i `error` događaje.

Čitanje veće količine podataka

Prethodni primer je ilustrovao čitanje jednog podatka iz IndexedDB baze. Čitanje je podrazumevalo poznavanje ključa podatka koji želimo da pročitamo. Pored takvog pristupa, vrlo česta je praksa čitanje veće količine podataka odjednom. To se može postići korišćenjem metode `getAll()`.

Metoda getAll()

Metoda `getAll()` omogućava čitanje svih ili dela podataka iz nekog objektnog skladišta. Njena sintaksa je sledeća:

```
getAll([query], [count])
```

Metoda `getAll()` može da prihvati dva parametra, pri čemu su oba neobavezna. Kada se pozove bez parametara, metoda `getAll()` obavlja čitanje svih objekata unutar objektnog skladišta. Dva ulazna parametra se mogu koristiti da dodatno filtriraju i ograniče objekte koji će biti pročitani. Prvim parametrom moguće je navesti seriju ili opseg ključeva čiji objekti će biti pročitani, dok je drugim parametrom moguće ograničiti ukupan broj objekata koji će biti pročitan.

Pre nego što pređemo na pisanje koda za čitanje svih objekata iz jednog skladišta, potrebno je da u bazu upišemo nekoliko različitih objekata:

```
let students = [{
    usi: "v43v2f",
    name: "John Steel",
    email: "email@mysite.com",
    age: 22
}, {
    usi: "3d23fv",
    name: "Ben Lord",
    email: "ben@mysite.com",
    age: 26
}];

students.forEach(function (student) {
    let request = studentsObjectStore.add(student);

    request.onsuccess = function () {
        console.log("Student added to the store",
request.result);
    };

    request.onerror = function () {
        console.log("Error", request.error);
    };
});
```

Kod za čitanje svih objekata iz `students` skladišta može da izgleda ovako:

```
        var studentsObjectStore =
db.transaction("students").objectStore("students");

        studentsObjectStore.getAll().onsuccess = function (event) {
            let studentsObj = event.target.result;

            studentsObj.forEach(function (student) {
                console.log(student.usi);
                console.log(student.name);
                console.log(student.email);
                console.log(student.age);
            });
        };
```

Primer unutar konzole proizvodi sledeći rezultat:

```
3d23fv
Ben Lord
ben@mysite.com
26
v43v2f
John Steel
email@mysite.com
22
```

Bitno je razumeti da se metodom `getAll()` dobijaju podaci spakovani unutar jednog niza. Zbog toga je u primeru za prolazak kroz takav niz iskorišćena metoda `forEach()`.

Napomena

Podaci unutar objektnih skladišta IndexedDB baze podataka uvek se sortiraju na osnovu vrednosti ključeva. Stoga se i prilikom čitanja oni dobijaju takvim redosledom, što često nije redosled kojim su oni upisani u skladište.

Korišćenje kursora za čitanje podataka

Veoma često, unutar skladišta IndexedDB baze postoji ogromna količina podataka. U takvim situacijama, čitanje svih podataka odjednom i njihovo smeštanje unutar niza može konzumirati veliku količinu interne memorije. Web pregledači definišu limite u takvim situacijama, pa veoma često može doći do zauzeća kompletne interne memorije koja je na raspolaganju jednom tabu web pregledača. Stoga je u slučaju čitanja velike količine podatka potrebno koristiti nešto drugačiji pristup, koji podrazumeva upotrebu kursora (*cursor*).

Kursor omogućava da se velika količina podataka čita zapis po zapis, što predstavlja znatno optimalnije rešenje iz ugla performansi.

Upošljavanje kursora postiže se metodom `openCursor()`.

Metoda openCursor()

Metoda `openCursor()` ima sledeću sintaksu:

```
openCursor([query], [direction]);
```

Metoda `openCursor()` može imati dva neobavezna parametra.

- `query` – jedan ili više ključeva kojima je moguće definisati koji će zapisi biti pročitani
- `direction` – smer u kome će se obavljati čitanje podataka; vrednost `next` definiše da će čitanje biti obavljeno počevši od ključa sa najmanjom vrednošću; to je i podrazumevana vrednost; kada se navede vrednost `prev`, čitanje se obavlja počevši od zapisa sa ključem najveće vrednosti

Kada se metoda `openCursor()` pozove bez parametara, dobija se kursor koji će čitati sve zapise unutar jednog skladišta.

Primer upotrebe kursora za čitanje svih podataka iz `students` skladišta može da izgleda ovako:

```
        var studentsObjectStore =
db.transaction("students").objectStore("students");

        studentsObjectStore.openCursor().onsuccess = function (event) {

            var cursor = event.target.result;

            if (cursor) {
                let studentObj = cursor.value;
                console.log(studentObj.usi);
                console.log(studentObj.name);
                console.log(studentObj.email);
                console.log(studentObj.age);
                cursor.continue();
            }
        };
```

Kod za čitanje podataka korišćenjem kursora podseća na onaj u kome se čitanje obavlja korišćenjem metode `getAll()`. Ipak, postoji jedna značajna razlika koju je potrebno da uvidite. Nakon aktiviranja funkcije `onsuccess`, unutar promenljive `cursor` biće smeštena referenca na objekat kursora. Unutar objekta kursora, jedan za drugim će biti učitavani zapisi iz skladišta baze podataka. Ipak, tako nešto se ne obavlja automatski, već smo u obavezi da, kada završimo sa jednim zapisom, pozovemo `countinue()` metodu kursor objekta i na taj način pomerimo kursor za jedno mesto unapred, odnosno na sledeći zapis iz skladišta. Kada u skladištu više nema podataka koje bi kursor pročitao, njegova vrednost postaje `undefined`, pa je upravo zbog toga referenca na ovaj objekat postavljena kao uslov if bloka. Kada više ne bude bilo zapisa, promenljiva `cursor` će dobiti vrednost `undefined`, pa se ni if uslovni blok neće izvršiti. U svakoj drugoj situaciji, `cursor` će imati vrednost različitu od `undefined`, pa se u takvim situacijama unutar if uslovnog bloka obavlja ispisivanje podataka objekata koji su pročitani iz baze. Za pristup podacima zapisa, kursor poseduje svojstva `key` i `value`.

Indeksi

Nešto ranije je ilustrovan pristup za kreiranje šeme baze podataka, što podrazumeva definisanje objektnih skladišta. Objektna skladišta se karakterišu nazivom i ključevima na osnovu kojih će biti moguće pristupati objektima skladišta. Ipak, prilikom kreiranja objektnih skladišta, pored definisanja naziva skladišta i ključeva objekata, moguće je definisati i indekse:

- indeksi omogućavaju da se, pored ključeva, podacima pristupa i na osnovu svojstava definisanih kao indeksi;
- indeksi omogućavaju da se definišu i ograničenja u vidu jedinstvenosti, koja vrednosti svojstava moraju da ispune.

Indeksi se kreiraju korišćenjem metode `createIndex()`.

Metoda createIndex()

Kreiranje indeksa se obavlja korišćenjem metode `createIndex()`. Njena sintaksa je:

```
objectStore.createIndex(name, keyPath, [options]);
```

Ulazni parametri imaju sledeću namenu:

- `name` – naziv indeksa
- `keyPath` – putanja do svojstva objekta nad kojim će biti kreiran indeks
- `options` – objekat kojim je moguće dodatno konfigurisati indeks koji se kreira; takav objekat može imati sledeće svojstvo:
    - `unique` – logička vrednost kojom se može definisati da vrednost svojstva indeksa svakog objekta u skladištu mora biti jedinstvena

Pošto su indeksi strukturalni elementi IndexedDB baza podataka, njih je neophodno definisati prilikom kreiranja šeme baze, odnosno unutar `onupgradeneeded` metode. U nastavku će biti ilustrovano kreiranje indeksa za svojstvo `name`, objekata koji predstavljaju studente. To će nam omogućiti da objektno skladište, pored ključa, pretražujemo i po vrednostima kreiranog indeksa, odnosno po vrednostima `name` svojstva.

```
request.onupgradeneeded = function (event) {

    db = event.target.result;

    var objectStore = db.createObjectStore("students", {
keyPath: "usi" });
    let index = objectStore.createIndex('name_idx', 'name');

};
```

Prikazanim kodom je definisano sledeće:

- kreiran je indeks koji će biti povezan sa svojstvom `name` objekata studenata;
- vrednosti svojstava `name` neće morati da budu jedinstvene, zato što nije definisan `options` objekat kod koga bi `unique` svojstvo imalo vrednost `true`.

Upravo prikazani kod za kreiranje indeksa može se upotrebiti ukoliko baza podataka još nije kreirana. Ukoliko se obavlja ažuriranje postojeće baze podataka, neophodno je definisati nešto drugačiju logiku.

Ažuriranje strukture baze podataka

Već više puta je rečeno da se struktura, odnosno šema IndexedDB baze podataka definiše unutar `onupgradeneeded` funkcije. Takva funkcija se aktivira ukoliko baza podataka ne postoji ili ukoliko je došlo do promene verzije, odnosno vrednosti parametra koji se prosleđuje `open()` metodi. Stoga je jedan od scenarija aktiviranja metode `onupgradeneeded` ažuriranje šeme baze podataka.

Ukoliko želite da ažurirate šemu baze podataka, prvo je neophodno da prilikom pozivanja metode `open()` prosledite vrednost uvećanu za jedan u odnosu na verziju trenutne šeme baze podataka:

```
var request = window.indexedDB.open("MyDatabase", 2);
```

Kao što vidite, prilikom poziva metode `open()` sada je kao drugi parametar prosleđena vrednost 2. Ovo će za efekat imati pozivanje funkcije `onupgradeneeded` prilikom sledećeg otvaranja stranice sa ovakvim kodom. Da je vrednost drugog parametra ostala 1, funkcija `onupgradeneeded` se ne bi aktivirala.

Pošto sada znamo da će se na ovaj način aktivirati `onupgradeneeded` funkcija, možemo preći na adaptaciju logike koja se unutar nje nalazi:

```
            request.onupgradeneeded = function (event) {
                db = event.target.result;
                var transaction = event.target.transaction;
                if (!db.objectStoreNames.contains('students')) {
                    var objectStore = db.createObjectStore("students", {
keyPath: "usi" });
                }

transaction.objectStore('students').createIndex('name_idx', 'name');
            };
```

Unutar `onupgradeneeded` funkcije definisano je nekoliko izmena. Prvo, kreiranje `students` objektnog skladišta sada će biti obavljeno samo ukoliko to skladište već ne postoji. To je postignuto dodavanjem jednog uslovnog bloka kojim se proverava da li skladište `students` postoji ili ne.

Nakon koda za eventualno kreiranje objektnog skladišta, definisana je i naredba za kreiranje indeksa. Objektno skladište nad kojim se kreira indeks sada je dobavljeno na nešto drugačiji način, pošto ne možemo da znamo da li će skladište biti kreirano ili već postoji. Stoga se do reference na objektno skladište dolazi korišćenjem implicitne transakcije koja postoji unutar `onupgradeneeded` funkcije.

Nakon kreiranja indeksa, on se praktično može iskoristiti prilikom čitanja, odnosno pretrage podataka:

```
        var studentsObjectStore =
db.transaction("students").objectStore("students");

        var index = studentsObjectStore.index("name_idx");

        index.get("Ben Lord").onsuccess = function (event) {
            console.log("Ben Lord's USI is " + event.target.result.usi);
        };
```

Na ovaj način obavlja se sledeće:

- kreira se transakcija za čitanje objektnog skladišta `students`
- dolazi se do reference na indeks `name_idx`
- indeks se koristi za čitanje jednog zapisa, po vrednosti `name` svojstva

Primer će unutar pregledača proizvesti sledeći ispis:

```
Ben Lord's USI is 3d23fv
```

Rezime

- Baza podataka je kolekcija strukturiranih informacija.
- IndexedDB je baza podataka koja postoji unutar web pregledača i na raspolaganju je web aplikacijama koje je mogu koristiti za čuvanje velike količine podataka na klijentu.
- IndexedDB je objektno orijentisana baza podataka, u kojoj se podaci čuvaju u formi parova ključeva i vrednosti.
- IndexedDB ne koristi SQL, već obezbeđuje sopstvene mehanizme za rukovanje podacima koji su bliski objektno orijentisanom modelu JavaScripta.
- IndexedDB obezbeđuje rukovanje podacima kroz transakcije.
- IndexedDB podržava indekse.
- Korišćenje IndexedDB baze podataka započinje otvaranjem konekcije korišćenjem metode `open()`.
- Prilikom otvaranja baze podataka može doći do emitovanja događaja `upgradeneeded`, `success` i `error`.
- Događaj `upgradeneeded` se aktivira ukoliko se baza podataka otvara prvi put ili ukoliko se vrši njeno ažuriranje.
- Šema je struktura baze podataka, a kod IndexedDB baza ona se definiše kreiranjem objektnih skladišta.
- Objektno skladište je moguće definisati korišćenjem metode `createObjectStore()`.
- Obavljanje bilo koje operacije nad IndexedDB podacima zahteva postojanje transakcije.
- Transakcija se kreira korišćenjem metode `transaction()`.
- Za dodavanje podataka u IndexedDB bazu koristiti se metoda `add()`.
- Za brisanje podataka iz IndexedDB baze koristi se metoda `delete()`.
- Metoda `get()` se koristi za čitanje jednog zapisa iz objektnog skladišta IndexedDB baze.

- Metoda koja se koristi za ažuriranje podataka IndexedDB baze je `put()`.
- Metoda `getAll()` omogućava čitanje svih ili dela podataka iz nekog IndexedDB objektnog skladišta.
- Kursor omogućava da se velika količina podataka čita zapis po zapis.
- Indeksi omogućavaju da se, pored ključeva, podacima pristupa i na osnovu svojstava definisanih kao indeksi.