Lokalno spremište podataka u HTML5

Hamzić, Antonijo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonijo Hamzić

LOKALNO SPREMIŠTE PODATAKA U HTML5

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonijo Hamzić

Matični broj: 0016148996

Studij: Informacijski i poslovni sustavi

LOKALNO SPREMIŠTE PODATAKA U HTML5

ZAVRŠNI RAD

Mentor/Mentorica:

Prof.dr.sc. Dragutin Kermek

Varaždin, rujan 2024.

Antonijo Hamzić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad će istražiti lokalne spremište podataka u HTML5. Opisat će osobine i vrste lokalnih spremišta podataka kao što su Web Storage API i IndexedDB. Usporedit će vrste po njihovim prednostima i nedostacima kao i njihovim performansama. Također će rad uključivati usporedbu lokalnih spremišta s drugim metodama pohrane podataka kao što su kolačići i sesijska spremišta. Također će uključivati JavaScript API za korištenje lokalnog spremišta podataka koji će nam omogućiti manipulaciju i pristup podacima u lokalnim spremištima podataka. Između ostalog potrebno će biti usporediti osobine web aplikacija s lokalnim i udaljenim spremištima podataka. Neki od prednosti web aplikacija s lokalnim spremištima podataka je da se mogu koristi podacima brže i izvan mreže. Također će rad proći kroz sinkronizaciju podataka lokalnog i udaljenog spremišta podataka. Pod tim poglavljem će se proći kroz različite tehnike sinkronizacija podataka. Važan aspekt rada je i praktični dio , a to je izrada same web aplikacije. Web aplikacija će morati raditi s lokalnim spremištem podataka.

Ključne riječi: lokalno spremište podataka, HTML5, pohrana podataka, web aplikacija, udaljeno spremište podataka

Sadržaj

1. Uvod

Kroz ovaj završni rad proći će se kroz teorijsku analizu i opis lokalnih spremišta u HTML-5 kao i praktična aplikacija koja će koristiti lokalna spremišta kao način spremanja podataka. Rad će proći kroz vrstu lokalnih spremišta kao i ostalih tipova spremanja koje postoje. Također će se kroz jedno poglavlje proći i kroz usporedbu lokalnih i udaljenih spremišta gdje će se vidjeti sve dobre i loše strane spremišta.

Praktični dio će se baviti implementacijom mrežne aplikacije (eng. web application). Glavna ideja aplikacije je praćenje vježbanja korisnika i spremanje tih podataka u lokalno spremište. Aplikacije će omogućiti korisniku intuitivno i zanimljivo praćenje svojih vježba. Također će se korisniku analizirati provedene vježbe preko koje može vidjeti svoj rad. Neke od funkcionalnosti aplikacije će biti dodavanje novih vježbi, pregled unesenih vježbi, uređivanje i brisanje vježbi, postavljanje i praćenje ciljeva, statistika i analitika, korisnički profil. Aplikacija će biti izrađena u HTML-u s CSS-om. JavaScript će se koristiti za korisničku stranu (eng. front-end) , a Node JS za klijentsku stranu (eng. back-end). Baza podataka će biti u SQLite-u.

Glavna motivacija odabira ove teme za završni rad je implementirati aplikaciju od nule koja će raditi i bez internet veze. Lokalno spremanje je dosta bitno kod izrade ovakvog tipa mrežnih aplikacija kako bi korisniku informacije bile brzo dostupne bez obzira na nekakav nenadan gubitak internetske veze.

2. Metode i tehnike rada

Praktični dio ovog završnog rada je izrada mrežne aplikacije tako da će biti potrebni neki alati i tehnike kako bi izrada aplikacije bila moguća.

Kako bi se krenulo s izradom aplikacije prvo je potreban alat za izradu dijagrama. Za izradu dijagrama koristit će se Draw.io koji je besplatni online alat za crtanje raznoraznih dijagrama.

Prvo i osnovno za samu izradu aplikacije potrebno je imati neko razvojno okruženje. Za izradu ovog rješenja koristit će se Visual Studio Code od Microsofta. Razlog ovog odabira je ta što je to jedan od najkorištenijih okruženja za izradu mrežnih aplikacija. Visual Studio Code je uređivač koda koji nudi podršku za brojne programske jezike tako i za one koji su potrebni kod izrade aplikacije za ovaj završni rad.

Također potreban je alat za izradu baze podataka. Baza podataka će se izraditi u MySQL Workbenchu. MySQL Workbench je alat koji omogućuje intuitivno izrađivanje MySQL i SQLite baza podataka.

Programski jezici koji će se koristiti za samu izradu su standardni za izradu mrežnih aplikacije. To su HTML, CSS i JS. HTML (HyperText Markup Language) je standardni jezik za izradu mrežnih aplikacija (eng. web application). CSS (Cascading Style Sheets) je jezik za opis izgleda i formata mrežnih aplikacija. JS (JavaScript) je skriptni jezik koji omogućuje dinamičnu interaktivnost mrežnih aplikacija.

Također da bi se aplikacija mogla pokrenuti potreban je i Node.js. Node.js je okruženje koje omogućuje pokretanje JS koda na poslužiteljskoj strani.

3. JavaScript

JavaScript je programerski jezik koji omogućuje dinamički rad mrežnih aplikacija. Koristi se u kombinaciji s HTML-om i CSS-om kako bi mrežne aplikacije mogle raditi. JavaScript je jezik koji omogućuje rad s lokalnim spremištima podataka.

Bez JavaScripta ne bi moderne mrežne aplikacije mogle uopće funkcionirati. Za bilo kakvu dinamičnost aplikacije potreban je JavaScript. Bilo kakvo spremanje podataka ili na primjer promjena na stranici nekakvim klikom na gumb omogućuje JavaScript.

JavaScript se također koristi sučeljima za programiranje aplikacija (eng. Application Programming Interfaces) ili skraćeno API-ma. API su već implementirani kodovi koji omogućuju lakšu implementaciju programerima nekih stvari koje bi bilo dosta kompleksno i vremenski zahtjevno za implementirati. API-jevi se mogu gledati kao već gotove dijelove koji se mogu ubaciti u aplikaciju. Jako važan API koji se koristi u JavaScriptu je objektni model dokumenta (eng. Document Object Model) API. DOM je zaslužan za bilo kakvu dinamičnu promjenu na HTML-u ili CSS-u. Također lokalna spremišta su dio API-a. API za mrežnu pohranu (eng. Web Storage API) sadrži u sebi lokalna spremišta [4].

3.1. Karakteristike JavaScripta

3.1.1. Osnovna sintaksa

Kao i svi programski jezici i JavaScript ima svoju sintaksu. Kroz ovo poglavlje će se proći kroz par osnovnih sintaksa koji su bitni kod rada s JavaScriptom i kako bi se lakše razumjele kasnije sintakse i naredbe s radom s lokalnim spremištima.

Prvo što je bitno kod programskog jezika su njegove varijable i koje su njihove mogućnosti. JavaScript varijable mogu biti definirane samostalno ili s ključnim riječima var,let i const. Const koristimo kada ne želimo da se vrijednost varijable promjeni, dok let i var imaju mogućnost promjene. Var se može koristiti unutar iste funkcije, dok let i const unutar istog bloka [4]. Primjer deklaracije varijabla:

```
var a=1;
let b=2;
const c=3;
```

JavaScript kao i svi programski jezici ima svoje tipove podataka. To su string, number, bigint, boolean, undefined, null, symbol, object i array. Kod JavaScripta nije potrebno posebno deklarirati varijablu s tipom podataka, on sam detektira tip podataka ovisno o tome koja je vrijednost pridružena [4]. Ako se deklarira broj ili string prepoznat će ga kao takvog, na primjer a varijabla će biti int, a b string:

```
let a=1;
let b="Ovo je string";
```

Jako bitan tip podataka za JavaScript je JSON (JavaScript Object Notation).JSON se koristi u JavaScriptu kao kompleksniji tip podataka koji u sebi sadrži objekt za lakši prijenos podataka. On u sebi može sadržavati više podataka. Primjer JSON-a koji u sebi sadrži popis dva studenta:

```
{
"studenti":[
{"Ime":"Antonijo", "Prezime":"Hamzić", "godina":3},
{"Ime":"Marko", "Prezime":"Marić", "godina":1}
]
}
```

Postoje dvije vrste komentara u JavaScriptu. To su jednolinijski i višelinijski:

```
//jednolinijski komentar
/*
Višelinijski komentar
*/
```

Funkcije u JavaScriptu su blokovi koda koji se moraju pozvati da bi se izvršio. Funkcije se definiraju ključnom riječi function i u sebi mogu imati parametre i mogu vratiti neku vrijednost. Primjer funkcije koja prima vrijednost i vraća vrijednost za jedan višu:

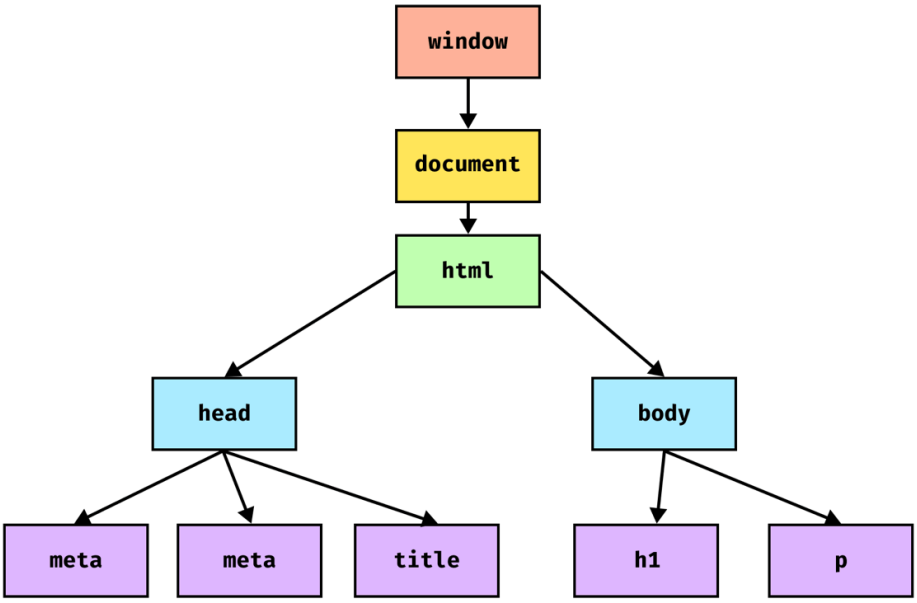
```
let a=primjer(1);

function primjer(b){
return b+1;
}
```

Kao i ostali programski jezici, JavaScript ima i izraze grananja i ponavljanja. Za grananje postoje if i switch, a za ponavljanje for,while, do-while, for-in i for-of. Primjer if izraza i for-in izraza u JavaScriptu:

```
if(a<10){
      poruka="Broj je manji od 10";
}

for(let a in studenti){
      imenaStudenata+=a.ime;
}
```

3.1.2. HTML DOM

Objektni model dokumenta predstavlja programsko sučelje za HTML. DOM je standard za promjenu i kreiranje HTML elemenata. U slučaju izrade mrežnih aplikacija DOM se može pristupiti JavaScriptom. JavaScript i DOM imaju puno mogućnosti. Neke od važniji i koji će se koristiti u implementaciji rješenja ovog završnog rada su manipulacija HTML objektima, CSS-om, stvaranje animacija, slušatelji događaja (eng. event listeners) [7].



Slika 1: Primjer DOM stabla [14]

Unutar DOM-a postoje objekti, metode i vrijednosti. Objekti predstavljaju HTML elemente, metode predstavljaju radnje koje možemo raditi na objektu, a vrijednosti nam služe za promjenu HTML objekata. Primjer objekta,metode i vrijednosti:

```
const popisStudenta = document.getElementById(imeStudenta);
popisStudenta.innerHTML= 'Antonijo';
```

Metoda koja se koristi getElementById traži na HTML-u element koji ima id imeStudenata, a vrijednost je innerHTML koja omogućuje zapis u HTML na to mjesto gdje je id koji je nađen.

DOM također omogućuje upravljanje događajima. Jedan od najčešćih upravljanja događajima je onClick koji omogućuje korisniku da pri kliku na određeni element (najčešće gumb) napravi neku aktivnost (najčešće pozivanje funkcije). Primjer gumba koji zove funkciju:

```
<button id="gumb" onClick="prikaziStudente()">Student</button);
```

Slušatelji događaja kao što sama riječ kaže slušaju i čekaju događaje te reagiraju na njih. Slušateljima događaja možemo zamijenit i neke od upravitelja događaja. Primjer slušatelja događaja koji se pokreće kada se pritisne gumb u prethodnom primjeru.

```
gumb.addEventListener('click', () =>{
const popisStudenta = document.getElementById(imeStudenta);
popisStudenta.innerHTML= 'Antonijo';
});
```

3.1.3. Asinkroni JavaScript

Asinkroni JavaScript je jako bitan kod dohvaćanja podataka. Naime služi kada se mora upravljati operacijama koje se ne mogu izvršiti u danom trenutku. Najčešće te operacije su dohvaćanje podataka sa udaljenog poslužitelja ili čitanja datoteka. Bez asinkronog programiranja ovakve akcije bi zahtijevale pauziranje cijelog programa dok se ne izvrše [4]. Dok asinkronim programiranjem ne stopiramo program kako bi dohvatili podatke.

Prvi primjer korištenja asinkronog programiranja je korištenje funkcije povratnog poziva (eng. callback). Funkcija povratnog poziva je funkcija koja se prosljeđuje drugoj funkciji kao argument i izvršava se tek kada ta funkcija izvrši svoj zadatak [13]. Primjer funkcije povratnog poziva:

```
function student(ime, callback){
      console.log(ime);
      callback();
}

function poruka(){
      console.log(' je student');
}

student('Antonijo', poruka);
```

Kako se funkcije povratnog poziva mogu brzo zakomplicirati i uvesti programera u probleme, uveden je objekt Promise. Promise je objekt koji predstavlja ishod asinkrone operacije. Mogu biti uspjeh (eng. resolved) i neuspjeh (eng. rejected). Primjer Promisa:

```
let promise = new Promise ((resolve, reject) => {
      let uspjeh = true;

      if (uspjeh){
            resolve("Uspjeh);
      } else{
            reject("Neuspjeh");
      }
});
```

Zadnja verzija JavaScript asinkronog programiranja je uvela nove ključne riječi. To su async i await. Ovaj način puno olakšava preglednost i način pisanja koda. Async označava funkcije koje vraćaju Promise, dok await odgađa izvršavanje funkcije dok se ne dobije Promise. Primjer korištenja async i awaita u dohvaćanju podataka o studentima:

```
async function dajStudente(){
	let parametri = {method : "GET");
	let odgovor = await fetch("baza/studenti",parametri);
	if(odgovor.status==200){
		let podaci=await odgovor.text();
		podaci = JSON.parse(podaci);
	}
}
```

3.2. Web API

JavaScript Web API predstavlja skup već gotovih funkcija koje omogućuje programerima lakšu izradu mrežnih aplikacija. Web API-ji dižu razinu funkcionalnosti, dinamičnosti i interaktivnosti mrežnih aplikacija. Postoji dosta vrsta Web API-ja, no najvažniji i oni koji će se koristiti u ovom završnom radu su DOM API, Fetch API, Canvas API i Web Storage API.

Kroz DOM se već prošlo u prethodnom poglavlju tako da nije bitno opet ponavljati isto. DOM API omogućava korištenje DOM-om kako bi se manipuliralo HTML-om. Najkorišteniji dio DOM API-ja je document.getElementById preko kojeg se dobiva elementi HTML-a preko njihovog ID-ja.

Fetch API omogućava jednostavan rad sa HTTP zahtjevima i odgovorima. Pomoću njega je moguće dohvaćati, mijenjati i dodavati podatke sa udaljenih poslužitelja. Fetch API se služi asinkronim programiranjem kako bi dobio podatke. Primjer jednostavnog dohvata iz prošlog poglavlja:

```
let odgovor = await fetch("baza/studenti",parametri);
```

Canvas API omogućuje crtanje i izradu grafika zajedno sa <canvas> elementom. Canvas API ima dosta široku primjenu, s pomoću njega je moguće napraviti od običnih grafova i slika, pa sve do izrade igara ili potpuno interaktivnih grafičke aplikacije. Primjer crtanja plavog pravokutnika sa Canvas API-jem:

```
let canvas = document.getElementById('canvas');
let kontekst = canvas.getContext('2d');
kontekst.fillStyle = 'blue';
kontekst.fillRect(10, 10, 150, 100);
```

I zadnji najbitniji za ovaj završni rad API je Web Storage API. Web Storage API pruža jednostavno spremanje podataka na korisnikovom pregledniku. Više o Web Storage API-ju će biti rečeno u sljedećem poglavlju.

4. Lokalna spremišta

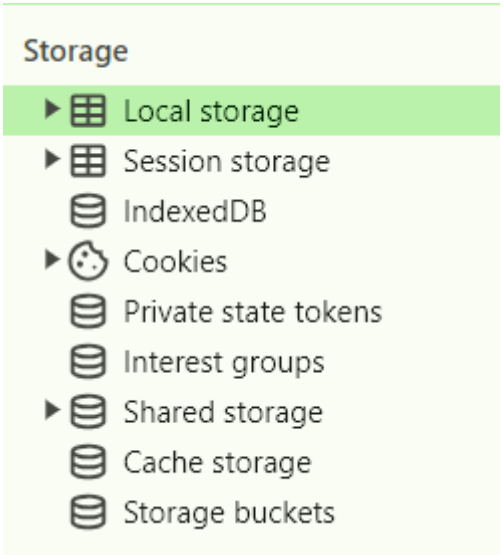
Kako bi bilo kakva aplikacija imala uspješan rad potrebne su neke vrste spremanja podataka, tako i mrežne aplikacije. Različiti su načini spremanja podataka, međutim dvije najvažnije podjele su lokalna i udaljena spremišta. Kao što same riječi govore lokalna spremišta spremaju podatke na računalo korisnika uz nadležnost web preglednika, dok udaljena spremišta spremaju podatke na udaljeni poslužitelj [3].

Lokalna spremišta uključuju baze podataka, datoteke ili neki drugi način spremanja koji je dostupan odmah na korisničkom uređaju. Lokalna spremišta omogućuju rad aplikacije i bez pristupa stalnoj internetskoj vezi. Zbog toga se koriste najviše na mobilnim i stolnim aplikacijama kojim ne treba stalna veza internetu. Međutim tema ovog rada su HTML5 lokalna spremišta te će se baviti lokalnim spremištima u mrežnim aplikacijama. Lokalna spremišta u mrežnim aplikacijama često uključuju tehnologije koje spremaju podatke direktno u korisnikov preglednik.

4.1. Vrste lokalnih spremišta

Kao što je u prošlom poglavlju bilo rečeno mrežne aplikacije imaju više vrsta lokalnih spremišta. Najvažniji će biti obrađeni kroz ovo poglavlje. Prvo i osnovno postoje dva člana Web Storage API-ja. Web Storage API je način spremanja podataka lokalno koji se pojavljuje u HTML5. Prije HTML5 lokalni podaci su se isključivo spremali u kolačićima koji će isto biti objašnjeni unutar ovog poglavlja.

Članovi Web Storage API-ja su Local Storage (lokalno spremište) i Session Storage (sesijsko spremište). Glavna razlika ta dva spremišta je to da sesijsko spremište briše podatke kada se preglednik ugasi, dok lokalno spremište i održava podatke neovisno o korisnikovom pregledniku. Također postoji i IndexedDB koji u sebi koristi indeksiranu bazu podataka za pohranu velikih količina podataka.



Slika 2: Vrste lokalnih spremišta unutar Google Chrome konzole [vlastita izrada]

4.1.1. Local Storage

Local storage je član Web Storage API-ja te je način lokalnog spremanja podataka. Najvažnija stvar kod njega je da se podaci spremaju u korisnikov preglednik i ostaju spremljeni iako korisnik ugasi preglednik. Jedini način da se podaci izbrišu iz lokalnog spremnika je taj da korisnik sam izbriše podatke ili da se podaci izbrišu kroz kod. Lokalno spremište i rad sa njim se može definirati unutar HTML5 i JavaScripta. Lokalno spremište sprema podatke pomoću ključa (eng. key) i vrijednosti (eng. value). Ograničenje količine podataka koje se mogu zapisati su u većini slučaja do 10 MB iako to zna ovisiti o korisnikovim postavkama ili postavkama samog preglednika [2]. Jedan primjer korištenja lokalnog spremišta je YouTube. YouTube u lokalno spremište sprema korisničke preferencije kao što su kvaliteta videa ili glasnoća zvuka.

Kao što je rečeno lokalno spremište sprema podatke pomoću ključa i vrijednosti međutim te vrijednosti jedino mogu biti u tipu podataka string te može onemogućiti korisniku unos neke od drugih tipova podataka. Također podaci se spremaju u čistom tekstu te se teže time manipuliraju podaci za razliku od na primjer JSON oblika, koji bi omogućio lakši prolazak kroz podatke.

Lokalno spremište ima četiri metode koje se koriste za rad sa njime. To su postavljanje (eng. set), dohvaćanje (eng. retrieve), uklanjanje (eng. remove) i čišćenje (eng. clear) [1].

Postavljanje služi za postavljanje vrijednosti u lokalno spremište. To jest dodaje novi element u samo spremište. Da bi se dodao element mora se definirati vrijednost kao i ključ podatka. Kako ova funkcija samo postavlja vrijednost, ne vraća povratnu vrijednost. Jedina moguća pogreška je QuotaExceededError koja se javlja ako je lokalno spremište popunjeno. Linija koda koja to omogućuje je:

```
localStorage.setItem(ključ,vrijednost);.
```

Dohvaćanje služi za dohvat vrijednosti koja se može koristiti. Da bi se dohvatio element mora se znati ključ elementa. Povratna vrijednost koju ova funkcija vraća je zapis uz određeni ključ kao string. Ova funkcija nema mogućnosti pogreške. Ako nema zapisa vraća se prazna (eng. null) vrijednost. Linija koda koja to omogućuje je :

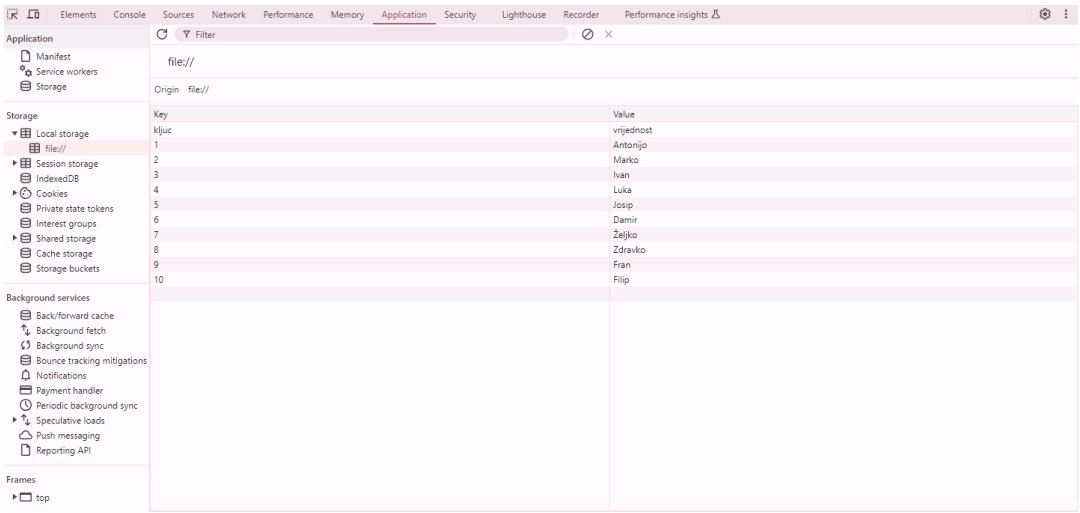
```
localStorage.getItem(ključ);.
```

Uklanjanje služi za micanje vrijednosti koju je potrebno maknuti. Da bi se maknuo element mora se znati ključ kako bi se izbrisao. Ova funkcija ne vraća povratnu vrijednost te ne baca pogreške. Linija koda koja to omogućuje je:

```
localStorage.removeItem(ključ);
```

Čišćenje služi za brisanje cijelog lokalnog spremišta. Kao i kod micanja vrijednosti ni ova funkcija nema pogreške i povratne vrijednosti. Linija koda koja to omogućuje je :

```
localStorage.clear();.
```



Slika 3: Primjer upisa u lokalno spremište unutar Google Chrome konzole [vlastita izrada]

4.1.2.Session Storage

Još jedan član Web Storage API-ja je Session storage. Session storage je način lokalnog spremanja podataka koji podatke sprema sesijski. Sesijsko spremanje podataka je način spremanja koje briše podatke nakon isteka sesije [1]. To znači da će se korisnički podaci izbrisati nakon svakog zatvaranja preglednika ili do isteka same sesije ako je to negdje definirano. Također razlika kod sesijskog spremanja i lokalnog je ta da se kod sesijskog podaci spremaju specifično za tu trenutno otvorenu karticu. Na primjer ako imamo spremljene podatke unutar jedne kartice i otvorimo novu karticu iako je iste domene, sesijski podaci će biti različiti. Naravno kao i kod lokalnog spremanja i podaci iz sesijskog spremanja se mogu brisati kroz kod. Međutim tu završavaju različitosti između sesijskog i lokalnog spremanja. Jedan od primjera korištenja sesijskog spremanja podataka je društvena mreža Instagram. Instagram sprema podatke o autentifikaciji u svoje sesijsko spremište.

Kao i kod lokalnog spremišta, sesijsko se može definirati unutar HTML5 i JavaScripta. Također način spremanja podataka je isti. Podaci se spremaju pomoću ključa i vrijednost [1] i. I ograničenje veličine podataka je isti kao i kod lokalnog, do 10 MB ovisno o korisnikovim postavkama ili postavkama preglednika.

Metode za rad sa sesijskim spremištem su također iste. To su postavljanje, dohvaćanje, uklanjanje i čišćenje. Metode rade iste stvari samo u sesijskom spremištu.

Postavljanje dodaje novi element u spremište, pomoću ključa i vrijednosti. Linija koda koja to omogućuje:

```
sessionStorage.setItem(ključ,vrijednost);.
```

Dohvaćanje dohvaća element iz spremišta, pomoću ključa. Linija koda koja to omogućuje:

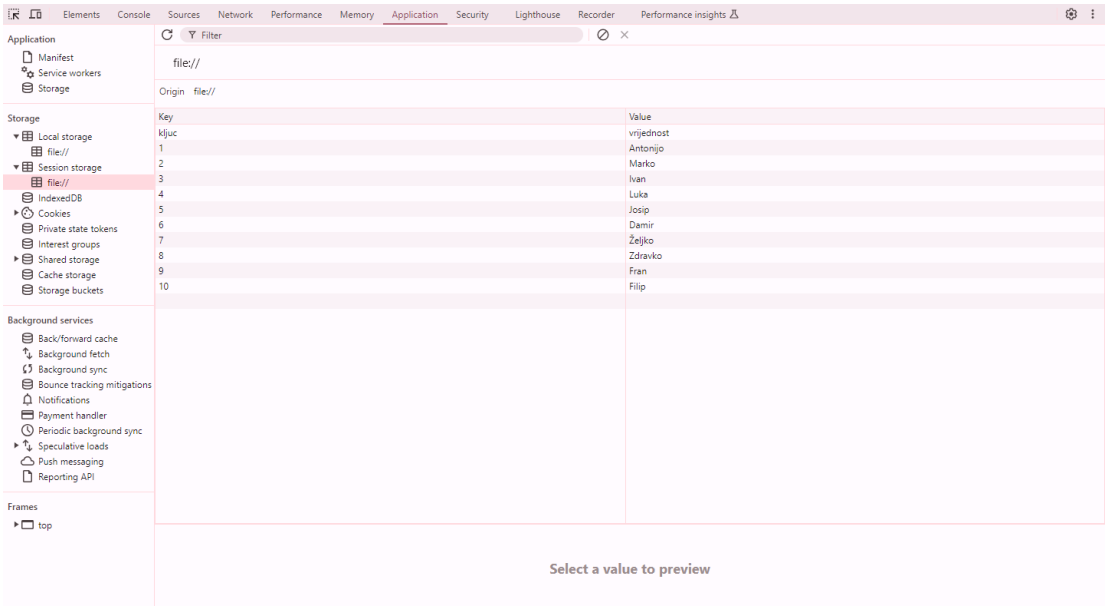
```
sessionStorage.getItem(ključ);.
```

Uklanjanje uklanja element iz spremišta, pomoć ključa. Linija koda koja to omogućuje:

```
sessionStorage.removeItem(ključ);
```

Čišćenje uklanja sve elemente iz spremišta. Linija koda koja to omogućuje:

```
sessionStorage.clear();
```



Slika 4: Primjer upisa u sesijsko spremište unutar Google Chrome konzole [vlastita izrada]

4.1.3. IndexedDB

Indexed Database API je noviji način za spremanje podataka lokalno. Velika razlika u odnosu na druge načine spremanja je ta da IndexedDB ima dosta veliko ograničenje veličine za spremanje podataka što omogućava bolji rad mrežne aplikacije bez Internet veze. Indexed Database API se koristi NoSQLom za rad s bazama podataka. NoSQL je skraćenica od not only SQL, ili ne samo SQL što znači da se ove baze podataka ne oslanjaju na isključivo tradicionalne relacijske metode nego proširuju metode sa nekim drugima radi boljih performansa. Primjer korištenja IndexedDB u današnjim mrežnim aplikacijama je Spotify. Spotify koristi IndexedDB za spremanje slika albuma i pjesama.

IndexedDB je zapravo baza podataka unutar preglednika. Što znači da može spremiti puno veće količine podataka od na primjer lokalnog spremišta. Indeksirane baze podataka kao i sve baze podataka čuvaju podatke. Podaci unutar njih mogu biti različitih vrsta što se razlikuje od drugih lokalnih spremišta. Mogu biti stringovi, brojevi, objekti i tako dalje. Također podupiru i transakcije koje omogućuju kvalitetniji i sigurniji rad. Baza podataka je indeksirana, što znači da se može efikasnije sortirati i pretraživati podaci pomoću indeksa. Također mogu se kreirati složeni upiti što još više pomaže u efikasnosti i boljoj korisnosti spremanja. Indeksirane baze podataka podupiru i asinkrono programiranje što znači da se ne blokiraju glavne dretve programa kada se radi s njima. Još jedna glavna stvar je ta da indeksirane baze podataka moraju upravljati verzijama. Što znači da sve promjene baze podataka moraju migrirati u drugu verziju.

Prva aktivnost koju je potrebno napraviti da bi se radilo sa indeksiranom bazom podataka je ta da se baza mora otvoriti. Za to je potrebno ime baze podataka i verzija baze podataka, u ovom primjeru ime baze će biti StudentDB, a verzija 1, zato što je to prva verzija koja se kreira.

```
let otvoriZahtjev = indexedDB.open(StudentDB, 1);
```

Zahtjev će vratiti objekt IDBOpenDBRequest koji može biti uspješan (eng. success), pogreška (eng. error) i potrebna nadogradnja (eng. upgradeneeded). Uspješan zahtjev govori da je baza uspješno otvorena, pogreška govori da nije, a potrebna nadogradnja da se baza može otvoriti, međutim verzija nije dobra.

Nakon što se baza podataka prvi put otvori ili nadogradi, sve operacije za kreiranje ili promjenu strukture baze moraju biti uključene unutar funkcije potrebna nadogradnja. Također će se unutar te funkcije pohraniti i objekt. Za razliku od ostalih baza podatka, podaci se ne spremaju u tablice, već u pohrane objekata (eng. object store). Svaki podatak u objektu mora imati ključ, slično kao i u lokalnim spremištima, međutim razlika je ta što ključ u objektu može biti u više oblika (broj, string, binarni). Greška koja se može dogoditi kod kreiranja objekata je ConstraintError. Ta greška se javlja kod pokušavanja kreiranja objekata s već postojećim imenom. Linija koda koja kreira objekt:

```
db.createObjectStore(ime[, opcijeKljuča]);
```

Ime označuje ime koje će objekt nositi, a opcije ključa je objekt u kojem se definira kako će ključ izgledati. U opcijama ključa može biti keyPath koji definira gdje će se ključ nalaziti i autoIncrement koji definira hoće li ključ biti samostalno inkrementiran. Primjer prvog pokretanja baze sa kreiranim objektom:

```
otvoriZahtjev.onupgradeneeded = function(event) {
let db = otovriZahtjev.result;
let studenti = db.createObjectStore("studenti", { keyPath: "id", autoIncrement: true });
};
```

Brisanje baze podataka je također vrlo jednostavno te je samo potrebno ime baze podataka. Kao i kod kreiranja baze ova funkcija vraća IDBOpenDBRequest. Primjer brisanja baze:

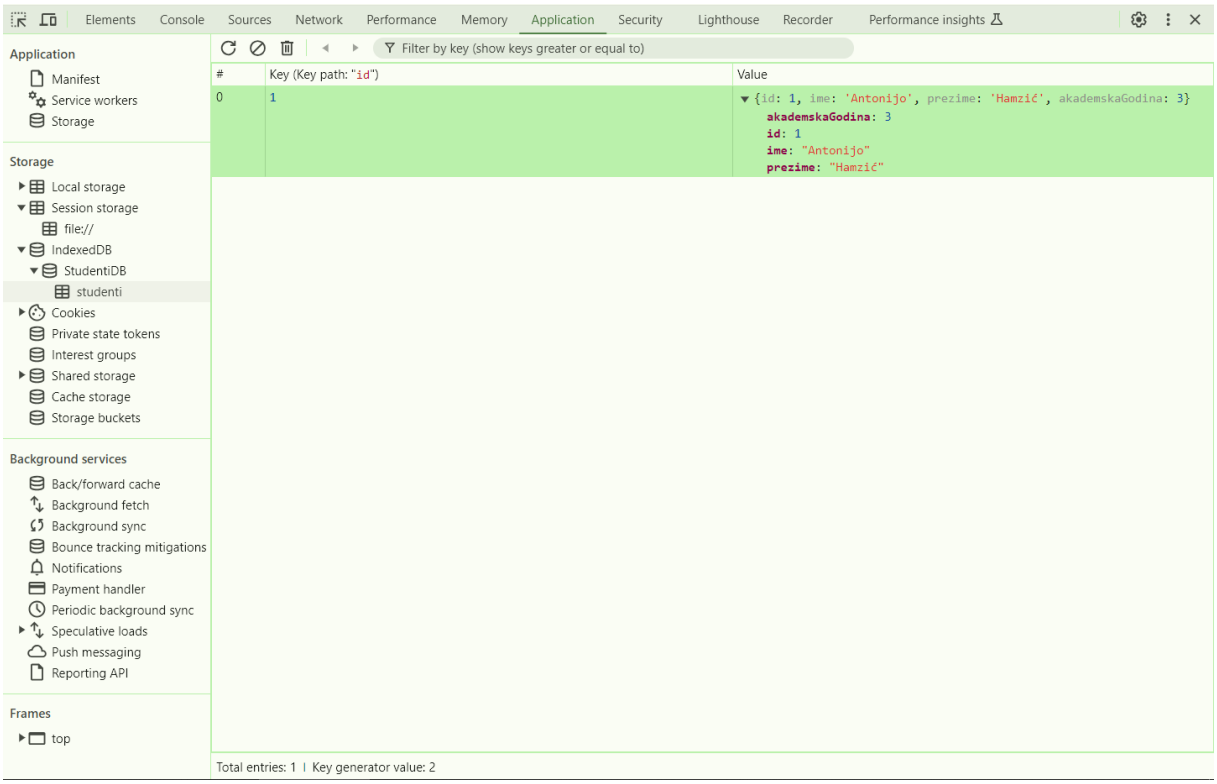
```
let izbrišiZahtjev = indexedDB.deleteDatabase(ime);
```

Da bi se podaci dodali linije koda koje su zaslužne za dodavanje podataka moraju se ukomponirati u funkciju uspješnog otvaranja baze. Također mora se uvesti i transakcija kako bi se podaci zapravo upisali. I za kraj zbog hvatanja grešaka mogu se dodati i dvije funkcije za praćenje uspješnosti zapisa u bazu. Kao i kod kreiranja objekata može se pojaviti ConstraintError. Primjer dodavanja podataka:

```
otvoriZahtjev.onsuccess = function(event) {
let db = otvoriZahtjev.result;
let transakcija = db.transaction("studenti", "readwrite");

let student = {
     id: 1,
     ime: Antonijo,
     prezime: Hamzić,
     akademskaGodina: 3
};
let zahtjev = studenti.add(student);

zahtjev.onsuccess = function() {
     console.log("Student uspješno dodan.");
};

zahtjev.onerror = function(event) {
     console.error("Greška prilikom dodavanja studenta:,
event.target.errorCode);
};
```

Također mogu se i objekti brisati:

```
db.deleteObjectStore('studenti');
```



Slika 5: Primjer upisa u IndexedDB unutar Google Chrome konzole [vlastita izrada]

4.1.4. Kolačići

Kolačići (eng. Cookies) su također opcija lokalnog spremanja podataka, koja se javila prije HTML5. Glavna snaga kolačića je da preglednik šalje podatke nazad na poslužitelja. Ali međutim u usporedbi sa ostalim načinim spremanja, maksimalna veličina je do 4 KB ovisno o web mjestu [5].

Kolačići se koriste i danas u mrežnim aplikacijama, posebno kod odobravanja pristupa korisnika i tokena. Također dosta poznata korist kolačića je spremanje korisnikovih podataka i reklama koje će mu se prikazivati.

Kolačići se mogu kreirati, čitati i brisati.

Za kreiranja kolačića mora se upisati njegov naziv, vrijednost. Naziv kolačića će biti korisnik, vrijednost Antonijo Hamzić, a trajat će 7 dana.

```
let date = new Date();
date.setTime(date.getTime() + (7*24*60*60*1000)):
let rokTrajanja = "expires=" + date.toUTCString();
document.cookie = "korisnik=Antonijo Hamzić; " + rokTrajanja + "; path=/";
```

Tradicionalno brisanje kolačića ne postoji. Međutim ako se stavi datum koji je prošao unutar kolačića , kolačić prestaje postojati.

```
document.cookie = "korisnik=; expires= Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

4.2. Ostale vrste spremišta

Kako bi se mogla u budućem poglavlju usporediti lokalna spremišta s ostalim vrstama mora se proći i kroz te ostale vrste spremišta.

Dvije najvažnije vrste spremišta nakon lokalnih su udaljena i hibridna spremišta. Kao što i same riječi govore udaljena spremišta spremaju podatke u udaljenim spremištima kao što su poslužitelji i računalni oblaci (eng. cloud). Dok hibridna spremišta koriste i lokalno i udaljeno spremište kako bi spremili podatke.

4.2.1. Udaljena spremišta

Udaljena spremišta su spremišta koja podatke ne spremaju kod korisnika. Najvažnije dvije vrste udaljenog spremanja podataka su pohrana na strani poslužitelja (eng. Server-side Storage) i pohrana u računalnom oblaku (eng. Cloud Storage). Glavna prednost udaljenih spremišta je da mogu u sebi pohraniti veliku količinu podataka, kao i sama sigurnost da podaci nisu dostupni vanjskom korisniku [9].

Pohrana na strani poslužitelja se koriste u većini modernih aplikacija. Najčešće se podaci spremaju na udaljenom poslužitelju u relacijske baze podataka kao što su MySQL ili PostgreSQL. Osim relacijskih baza podataka popularne su i NoSQL baze podataka koje se koriste i u već spomenutom IndexedDB, a u slučaju pohrani na strani poslužitelja NoSQL baze podataka koje su popularne MongoDB i Firebase Firestore. Pošto su podaci spremljeni na udaljenom poslužitelju podaci su puno sigurniji i teže je nekom vanjskom korisniku doći do tih podataka. Bitno je i to da baze podataka omogućuju centralizirano upravljanje podataka, to jest svim podacima se može manipulirati iz jednog centralnog spremišta kojem se može pristupiti programima koji upravljaju bazama podataka kao što su DBeaver ili SSMS. Nedostatci ovakvog pristupa spremanja podatak su da je uvijek potrebna mrežna veza za pristup podacima, također pristup podacima može biti nešto sporiji od lokalnih spremišta.

Pohrana u računalnom oblaku je način spremanja podatka koji spremaju podatke u udaljene oblake koji su održavani od strane nuditelja te usluge. Neki od primjera pružatelja tih usluga su Google Cloud Storage i Azure Blob Storage. Pohrana u oblaku je zanimljiva jer nudi dosta veću dostupnost i mogućnosti skaliranja mrežnih aplikacija. Također usluge u oblaku mogu omogućiti dosta drugih korisnih stvari osim samog spremanja podataka. Također pohrana u oblaku se plaća ovisno o korištenju i veličini podataka. Tako da što je veći projekt to će rasti i sama cijena usluge.

4.2.2. Hibridna spremišta

Hibridna spremišta su tipovi spremišta koji kombiniraju lokalna i udaljena spremišta za rad s podacima. Hibridna spremišta su također dosta česta u izradi mrežnih aplikacija jer poboljšavaju performanse u usporedbi sa običnim udaljenim spremištima. Prvi primjer hibridnih spremišta su sinkronizacija klijent-poslužitelj (eng. Client-Server Synchronization). Također primjeri hibridnih spremišta mogu biti i API-ji kojim se dohvaćaju i pohranjuju podatci na poslužitelju, primjeri takvog načina spremanja su GraphQL i RESTful API.

Glavna ideja sinkronizacije klijent-poslužitelj je da se aplikacija može koristiti i izvan mreže. To je tip spremanja u kojem se kombiniraju lokalna i udaljena spremišta u kojem se podaci spremaju lokalno i sinkroniziraju sa poslužiteljom. To znači da kada je aplikacija spojena na internet vuče i sinkronizira podatka s udaljenog poslužitelja te ih sprema u neki tip lokalnog spremišta. Glavni primjer takvih mrežnih aplikacija su progresivne mrežne aplikacije (eng. Progressive web apps). Progresivne mrežne aplikacije su aplikacije koje funkcioniraju kao stolne aplikacije te se mogu i instalirati na korisničko računalo i raditi bez mrežne veze. Glavni problem takvog spremanja podatka je potreba za rješavanje konflikta između podataka. Naime podaci se stalno sinkroniziraju i postoji šansa da se lokalni podaci razlikuju od udaljenih zbog nekih ručnih promjena na bazi podataka od strane admina. Ako se to dogodi ručno se moraju mijenjati i rješavati konflikti između podataka.

Korištenje API-ja kao što su GraphQL i RESTful API za pohranu podatka je također jedna vrsta hibridnog spremanja. To je zato jer se često koristi lokalnim spremištima kako bi predmemorirali (eng. caching) podatke. GraphQL i RESTful API su API strukture koje omogućuju razmjenu podataka na klijentu i poslužitelju. Oni se koriste tako da klijent pošalje API zahtjev na poslužitelju te poslužitelj njemu odgovori ovisno o uspješnosti pozivanja. GraphQL omogućuje klijentu da specificira podatke koje želi dobiti koristeći fleksibilne upite. Dok RESTful koristi HTTP metode (GET, POST, PUT, DELETE) kako bi komunicirao sa poslužiteljom i dobio rezultate.

4.3. Usporedba lokalnih i udaljenih spremišta podataka

U prošlim poglavljima definiralo se što su to lokalna i udaljena spremišta te koje metode i funkcionalnosti imaju u sebi. Te zadatak ovog poglavlja je proći kroz glavne karakteristike i razlika između lokalnih i udaljenih spremišta.

4.3.1. Prednosti i nedostaci lokalnih spremišta

Najveća prednost lokalnih spremišta je sama brzina pristupa podacima. Podacima se relativno brzo pristupa zbog toga što su svi podaci lokalno spremljeni u pregledniku te nema potrebe za komunikacijom između drugih aplikacija ili poslužitelja.

Dakako isto je velika prednost posebno u usporedbi s udaljenim spremištima ta da su podaci dostupni i bez internetske veze. Tako da podaci mogu uvijek biti spremni za korištenje bez obzira na internetsku vezu i njezinu brzinu. To također poboljšava i brzinu samog pristupa podataka jer je neovisna o internetskoj brzini.

Kao što se vidjelo u prošlim poglavljima implementacija ovakvih spremišta je također vrlo jednostavna. Uz par linija koda mogu se spremati podatci, bilo to s lokalnim ili sesijskim spremanjem. Također i vrlo brzo se može napraviti i baza podataka unutar koda uz pomoć IndexedDB. Ta vrlo jednostavna i brza implementacija također ubrzava proces spremanja podataka. Tako da ne samo da su lokalna spremišta brža u spremanju i načinu rada, nego su brzi i u samoj implementaciji. Nisu potrebni nikakvi vanjski poslužitelji ili alati za izradu spremišta.

Naravno uz prednosti lokalnih spremišta dolaze i nedostaci. Jedan od nedostataka je dakako jako ograničen prostor za pohranu. Kod lokalnih spremišta se mora paziti koliko podataka se sprema kao i koliko će ti podaci bit veliki. Lokalna spremišta u većini slučaja imaju ograničenje količine podataka od otprilike 5MB [10]. Što je dosta malo za bilo kakvo ozbiljnije spremanja podataka.

Nije samo problem male mogućnosti spremanja veličine podataka. Također je problem i manjak mogućnosti kod spremanja tipa podataka. Naime lokalnim spremištima se mogu spremiti samo string vrijednosti, što smanjuje kvalitetu i mogućnosti same aplikacije.

Također iako je na papiru lokalno spremište brže, stvaraju se problemi kod dohvaćanja više podataka. Naime lokalna spremišta su sinkrona, što znači da se čeka završetak svake metode kako bi se pokrenula nova.

Lokalna spremišta također imaju problem sa sigurnošću. Sve funkcije su pohranjene unutar JavaScript koda, što znači da se može lakše pristupiti tim podacima i zloupotrijebiti ih. Tako da je važno za što se točno koristi lokalno spremište. Definitivno ne bi bilo pametno koristiti lokalna spremišta za spremanja nekih podataka koji se mogu zloupotrijebiti (kao što su na primjer lozinke).

4.3.2. Prednosti i nedostaci udaljenih spremišta

Glavna prednost udaljenih spremišta u usporedbi s lokalnim je veličina podataka koja se može pohraniti. U usporedbi s lokalnim spremištima koje mogu pohraniti samo 5MB, udaljena spremišta mogu pohraniti puno više (veličina memorije može ići i u više terabajta). Tako da su razlike u mogućnosti veličine podataka dosta velike.

Kao što je rečeno kod lokalnih spremišta oni mogu pohraniti samo string vrijednosti. Dok kod udaljenih spremišta postoje raznorazne mogućnosti spremanja tipova podatak ovisno o načinu na koju se korisnik odluči. To omogućuje novu dimenziju fleksibilnosti podataka koje se koriste pri izradi mrežnih aplikacija.

Također kod rada s podacima bitna je i sigurnost. Za razliku od lokalnih spremišta podataka, udaljena spremišta mogu biti implementirana s većom dozom sigurnosti. Ovisno o potrebama može biti bolje i lošije implementirana, međutim postoji ta mogućnost tako da je to jedna od velikih prednosti udaljenih spremišta. Dok kod lokalnih ne postoji mogućnosti za tako nešto.

Još jedna pozitivna stvar kod udaljenih spremišta je ta da su podaci dostupni na različitim uređajima i lokacijama. Dok kod lokalnih spremišta korisnici su ograničeni na jedan uređaj, u udaljenim spremištima ne postoji taj problem te se podaci mogu dijeliti međusobno između uređaja i korisnika,

Glavna prednost lokalnih spremišta je i glavna mana udaljenog spremišta. A to je potreba udaljenih spremišta za internetskom vezom. Kod udaljenih spremišta svaka komunikacija s podacima mora proći kroz internetsku vezu. Što može izazvati probleme u brzini dobivanja i spremanja podataka. Kod sporijih internetskih veza sporija je i komunikacija podataka, što nije bio problem kod lokalnih spremišta podataka.

Također nedostatak udaljenih spremišta je ovisnost o nečem trećemu. Ovisno o načinu rada sa udaljenim spremištima dolaze problemi na koje korisnik ne može utjecati. Ti problemi mogu biti kod poslužitelja ili pak ako je spremište tipa na računalnom oblaku koju održava treća strana.

4.3.3. Primjena u praksi

Nakon prolaska kroz prednosti i nedostataka lokalnih i udaljenih spremišta podataka, može se zaključiti da oba spremišta imaju svoje koristi u modernim mrežnim aplikacijama. Te je dosta bitno ispravno odlučiti koji način spremanja je potreban u kojoj aplikaciji.

Lokalna spremišta su bitna i bolja opcija kod nekih jednostavnih aplikacija. Aplikacije koje trebaju spremiti male količine podataka kao što su neke korisničke postavke ili pak neki privremeni podaci koje su korisniku trenutno potrebni. Kao što je rečeno mnogo puta kroz završni rad glavna primjena lokalnih spremišta uz spremanja privremenih podataka su aplikacije koje mogu raditi i bez mrežne veze. I to je najveća snaga i primjena takvih spremišta u praksi.

Udaljena spremišta su dakako dosta korištenija u praksi nego lokalna. Svaka moderna mrežna aplikacija mora koristiti udaljena spremišta kako bi mogla raditi. Udaljena spremišta imaju raznorazne primjene koje su potrebne mrežnim aplikacijama. Udaljena spremišta su nam potrebna za sinkronizaciju podataka između korisnika, bez udaljenih spremišta nije moguće imati više korisnika koji mogu međusobno komunicirati. Bez udaljenih spremišta nije moguće koristiti mrežnu aplikaciju sa istim postavkama i informacijama na dva različita uređaja. Bilo kakva aplikacija koja zahtjeva sigurnosne kopije i oporavak podataka mora se koristiti udaljenim spremištima.

Zaključak primjene u praksi je taj da svaka mrežna aplikacija je priča za sebe i ne postoji pravi odabir za svaku aplikaciju. Neke su jednostavne i stvarno ne trebaju udaljena spremišta, dok su druge toliko kompleksne da im je potrebna internet veza i ne trebaju nikakvo lokalno spremanje. Također uvijek postoji i hibridno rješenje, u kojem jedan dio se sprema lokalno kao što su korisničke postavke ili pak dio podataka da bi se mogla aplikacija koristiti i bez mreže. Dok drugi dio obavlja udaljeno spremište koje omogućuje aplikaciji normalan i brz rad dok ima internetske veze.

4.4. Sinkronizacija podataka

Kao što je već spomenuto dosta modernih mrežnih aplikacija koriste hibridne spremnike. To jest kombinaciju lokalnih i udaljenih spremišta podataka. Da bi bio moguć takav rad aplikacije potrebno je sinkronizirati podatke.

Sinkronizacija podataka je način na koji povezujemo lokalna i udaljena spremišta. Ne smije se dogoditi da se podaci razlikuju na lokalnim i udaljenim spremištima, naravno ako su povezani. Podaci moraju biti usklađeni kako bi aplikacija ispravno radila.

Sinkronizacija podataka je jako bitna u aplikacijama koje koriste lokalna spremišta za rad bez internetske veze. To znači da za te aplikacije je potrebno jako dobro odraditi sinkronizaciju inače aplikacije neće funkcionirati. Takve aplikacije u biti koriste udaljeno spremište za normalan rad sa internet vezom. Međutim imaju mogućnost rada bez internet veze di se udaljeni podaci mogu spremiti lokalno i pri ponovnom povezivanju ti podaci se moraju nazad spremiti udaljeno. Zato je bitna sinkronizacija koja se mora obaviti bez znanja krajnjeg korisnika.

4.4.1. Metode sinkronizacije podataka

Za sinkronizaciju podataka postoje raznorazne metode sinkronizacije. U ovom poglavlju će se proći kroz tri najvažnije sinkronizacije podataka. Postoje sinkronizacija u stvarnom vremenu, asinkrona sinkronizacija i skupna (eng. batch) sinkronizacija.

Sinkronizacija u stvarnom vremenu sinkronizira podatke lokalnih i udaljenih spremišta u istom trenutku u kojem se dogode. To znači da čim se neka promjena dogodi na lokalnom spremištu ta promjena se mora samostalno dogoditi i na udaljenom spremištu. Takva sinkronizacija je bitna kod aplikacija koje stalno razmjenjuju podatke između korisnika ili uređaja. Njima je nužno da su podaci brzo i samostalno sinkronizirani.

Asinkrona sinkronizacija je tip sinkronizacije podataka koji se događa s odgodom. To jest postoji vrijeme koje prođe da bi se podaci sinkronizirali. To znači da podaci nisu uvijek sinkronizirani. Podaci se mogu na razne načine asinkrono sinkronizirati. Mogu imati vremenski period kada se ti podaci sinkroniziraju, neku akciju koja pokreće proces sinkronizacije (kao na primjer prebacivanje iz izvanmrežnog načina u mrežni način) ili pak ručno korisnik može pokrenuti sinkronizaciju. Takva metoda je dobra za aplikacije sa izvanmrežnim radom, u kojima nije potrebno odmah sinkronizirati podatke i u kojima se podaci ne razmjenjuju stalno.

Skupna sinkronizacija je tip sinkronizacije udaljenih i lokalnih spremišta u kojem se sve promjene šalju u skupu. To znači da postoji period u kojem se sve promjene u lokalnim spremištima šalju na udaljeno spremište. Ova metoda je dobra kada korisniku nije bitna brzina i vrijeme sinkronizacija podataka. Bolja je za performanse aplikacije i smanjuje stalno opterećenje mreže zato što se svi podaci šalju u isto vrijeme i nema nepotrebne stalne komunikacije između lokalnih i udaljenih spremišta.

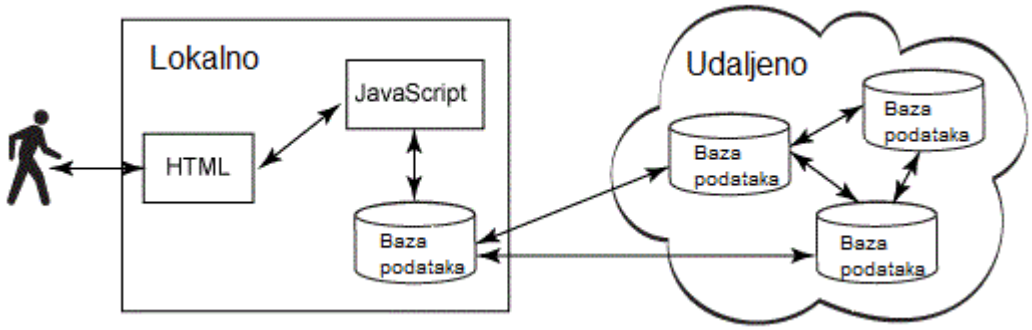
4.4.2. Tehnologije i alati sinkronizacije podataka

Kako bi se lakše izvela sinkronizacija podataka između lokalnih i udaljenih spremišta podataka mogu se koristiti različite tehnologije i alati. Sinkronizacija se može također i ručno izvesti. Dva primjera koja će se obraditi u sklopu ovog poglavlja su kombinacija PouchDB i CouchDB te Firebase Realtime Database.

Kombinacija PouchDB i CouchDB je primjer baze podataka koji omogućuje sinkronizaciju između lokalnih i udaljenih baza podataka. PouchDB i CouchDB su napravljeni kako bi sinkronizacija između spremišta bila što lakša. PouchDB služi za lokalno spremanje podataka i nalazi se na korisnikovom pregledniku. Dok je CouchDB udaljena baza podataka. PouchDB i CouchDB su NoSQL baze podataka koje spremaju svoje podatke u dokumente.

Nude dva tipa sinkronizacije. Prvo je naravno sinkronizacija u pravom vremenu, gdje se samostalno prenose podaci lokalno i udaljeno. Drugi način je onaj koji se događa kada se izgubi internetska veza. Naime ove baze podataka spremaju podatke lokalno i kada se vrati internet veza sinkronizira podatke nazad u udaljeno spremište.

Također unutar tih baza podataka se mogu lagano i pratiti promjene. U posebnoj listi koja se zove lista promjena (eng. changes feed) su popisi svih promjena koje su se dogodile na bazi podataka.



Slika 6: Sinkronizacija podataka PouchDB i CouchDB [15]

Firebase Realtime Database je alat za sinkronizaciju spremišta u pravom vremenu. Realtime Database je samo dio Firebasea. Firebase je Googleov alat u računalnom oblaku koji se koristi za mobilne aplikacije. Realtime Database sinkronizira podatke između aplikacije i oblaka. Također kao i PouchDB i CouchDB koristi se lokalnom pohranom kada uređaj nema internetske veze. Firebase Realtime Database se također koristi NoSQL kako bi spremao podatke. Pošto je NoSQL podaci se spremaju u JSON-u što omogućava laganu sinkronizaciju između lokalnih i udaljenih podataka.

Ovakvi načini sinkronizacije su dosta dobri i intuitivni. Omogućuje lakši i sigurniji razvoj aplikacija. Dok ručna sinkronizacija može biti teža i puna sigurnosnih promjena.

5. Aplikacija

5.1. Opis aplikacije

Glavni zadatak aplikacije TrackOut je praćenje vježba korisnika. Aplikacija je rađena u HTML-u, CSS-u i JavaScript-u. Korišten je nodeJS za poslužiteljsku stranu aplikacije te SQLite za bazu podataka.

Aplikacija se sastoji od sedam stranica. Kada se aplikacija pokrene prikazana je prvo stranica za prijavu. Na stranici za prijavu korisnik se može prijaviti ili se može prebaciti na stranicu za registraciju. Kada se korisnik uspješno prijavi preusmjeren je na glavnu stranicu.

Glavna stranica se sastoji od tri odlomka. Odlomak za unos vježba, odlomak za statistiku i odlomak za ciljeve.

Zatim postoji stranica statistike. Na njoj se nalaze današnja i sveukupna statistika korisnika te tri grafa koji prikazuju različite informacije o vježbanju.

Na stranici profila postoje dva odlomka. Jedan odlomak prikazuje osnovne informacije o korisniku koje se mogu urediti. Dok drugi odlomak prikazuje kalendar slika koje korisnik može mijenjati. Također na stranici profila može se promijeniti i slika profila koja se nalazi na svakoj stranici.

Na stranici ciljeva također postoje dva odlomka. Jedan odlomak prikazuje današnje vježbe odrađene i odrađene ciljeve. Također je moguće dodati nove ciljeve. Drugi odlomak prikazuje postignute kilaže koje je korisnik odradio.

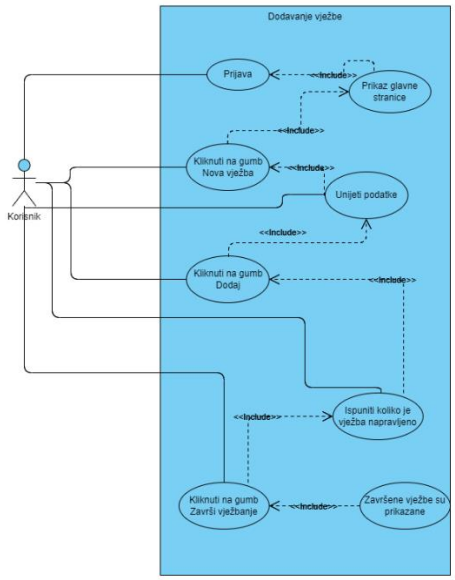
Zadnja stranica je stranica sa popisom vježba. Na stranici se nalaze glavne informacije o vježbama te je moguće dodavati svoje vježbe.

Stranica sve informacije iz baze podataka sprema u lokalno spremište. Kod svakog dohvata novih podataka prvo se provjerava postojanje tih podataka u lokalnom spremištu. Dok se informacije o prijavljenom korisniku spremaju u sesijsko spremište.

5.2. Dijagrami

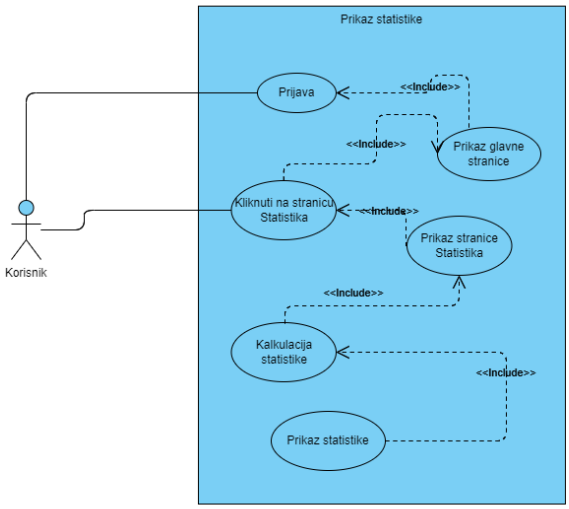
5.2.1. Dijagram slučajeva korištenja

Prvi dijagram slučajeva korištenja prikazuje kako korisnik komunicira sa sustavom kod dodavanja nove vježba.



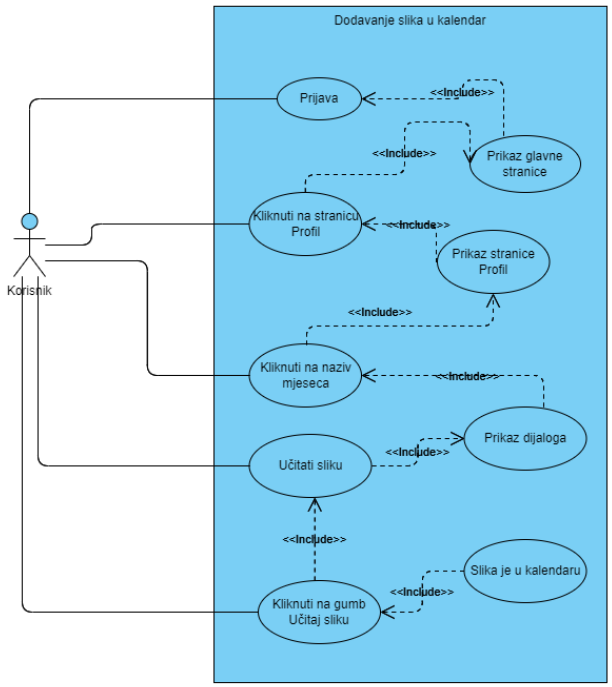
Slika 7: Dijagram slučaja korištenja Dodavanje vježbe [vlastita izrada]

Drugi dijagram slučajeva korištenja prikazuje kako korisnik komunicira sa sustavom kod prikaza statistike korisnika.



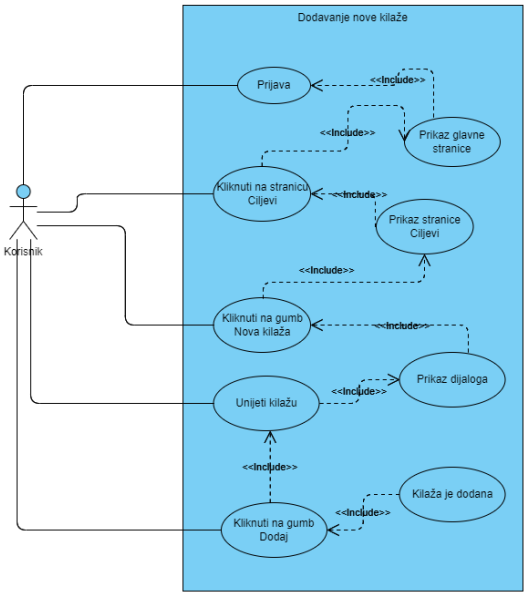
Slika 8: Dijagram slučaja korištenja Prikaz statistike [vlastita izrada]

Treći dijagram slučajeva korištenja prikazuje kako korisnik komunicira sa sustavom kod dodavanja novih slika u kalendar.



Slika 9: Dijagram slučaja korištenja Dodavanje slike u kalendar [vlastita izrada]

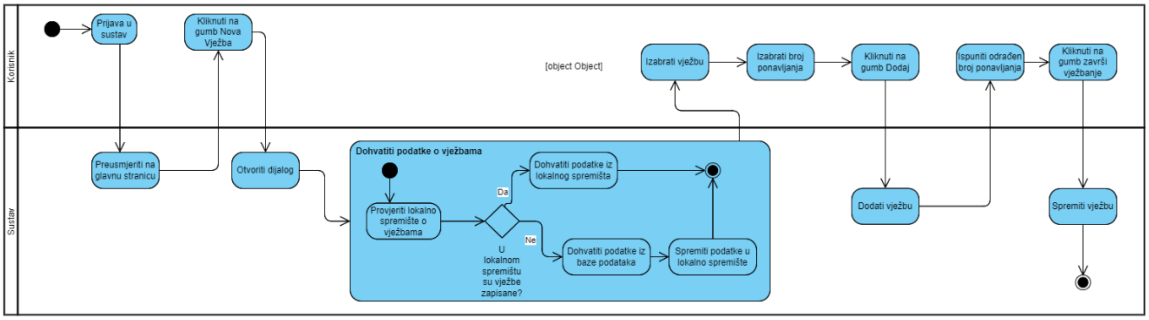
Četvrti dijagram slučajeva korištenja prikazuje kako korisnik komunicira sa sustavom kod dodavanja nove kilaže vježba.



Slika 10: Dijagram slučaja korištenja Dodavanje nove kilaže [vlastita izrada]

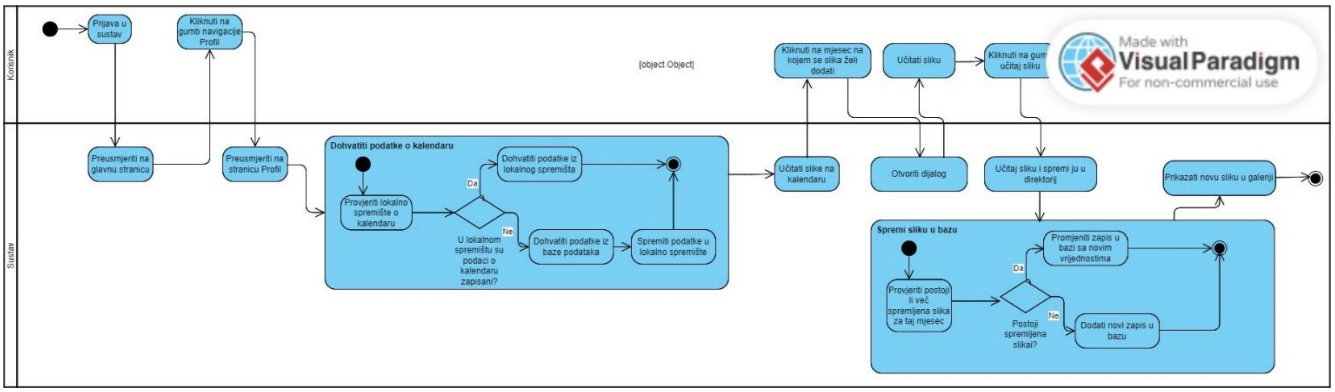
5.2.2. Dijagram aktivnosti

Prvi dijagram aktivnosti prikazuje kako se dodaje nova vježba. Dijagram prikazuje i kako sustav dohvaća podatke o vježbama.



Slika 11: Dijagram aktivnosti Dodavanje vježbe [vlastita izrada]

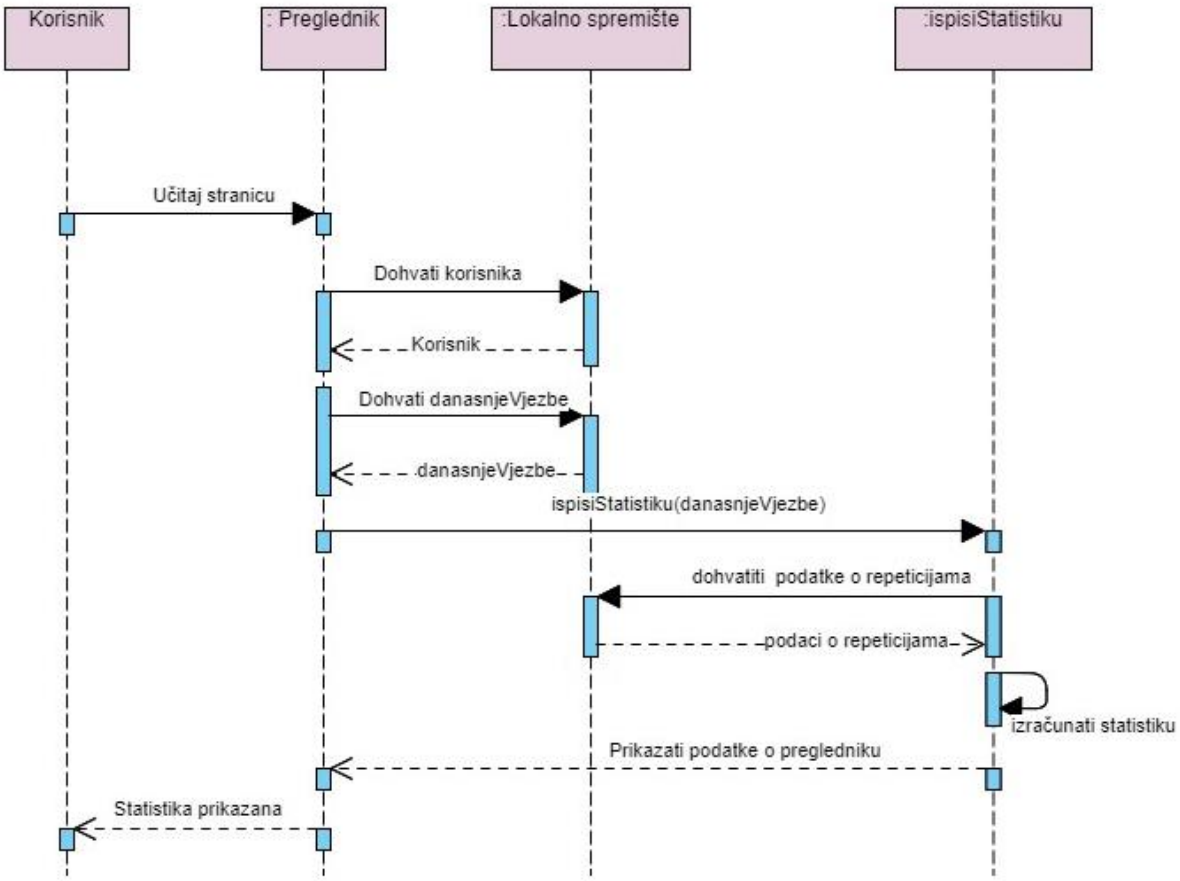
Drugi dijagram aktivnosti prikazuje kako se učitava slika na kalendaru. Dijagram prikazuje i kako sustav dohvaća podatke o vježbama i kako se slika sprema u bazu podataka.



Slika 12: Dijagram aktivnosti Dodavanje slike [vlastita izrada]

5.2.3. Dijagram slijeda

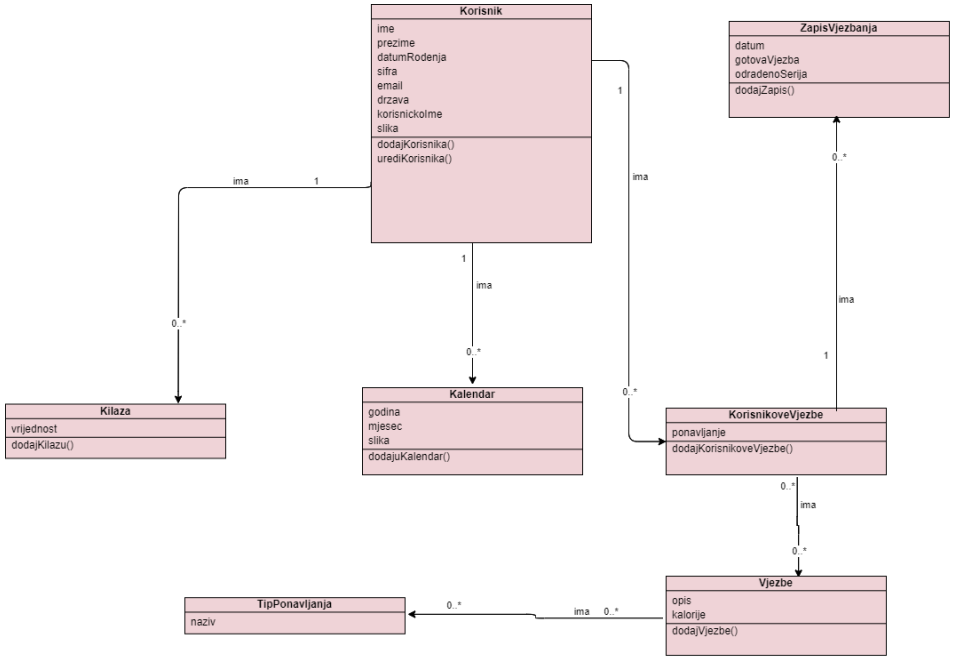
Ovaj dijagram slijeda prikazuje kako sustav ispisuje statistiku o vježbama.



Slika 13: Dijagram slijeda korištenja Prikaz statistike [vlastita izrada]

5.2.4. Dijagram klase

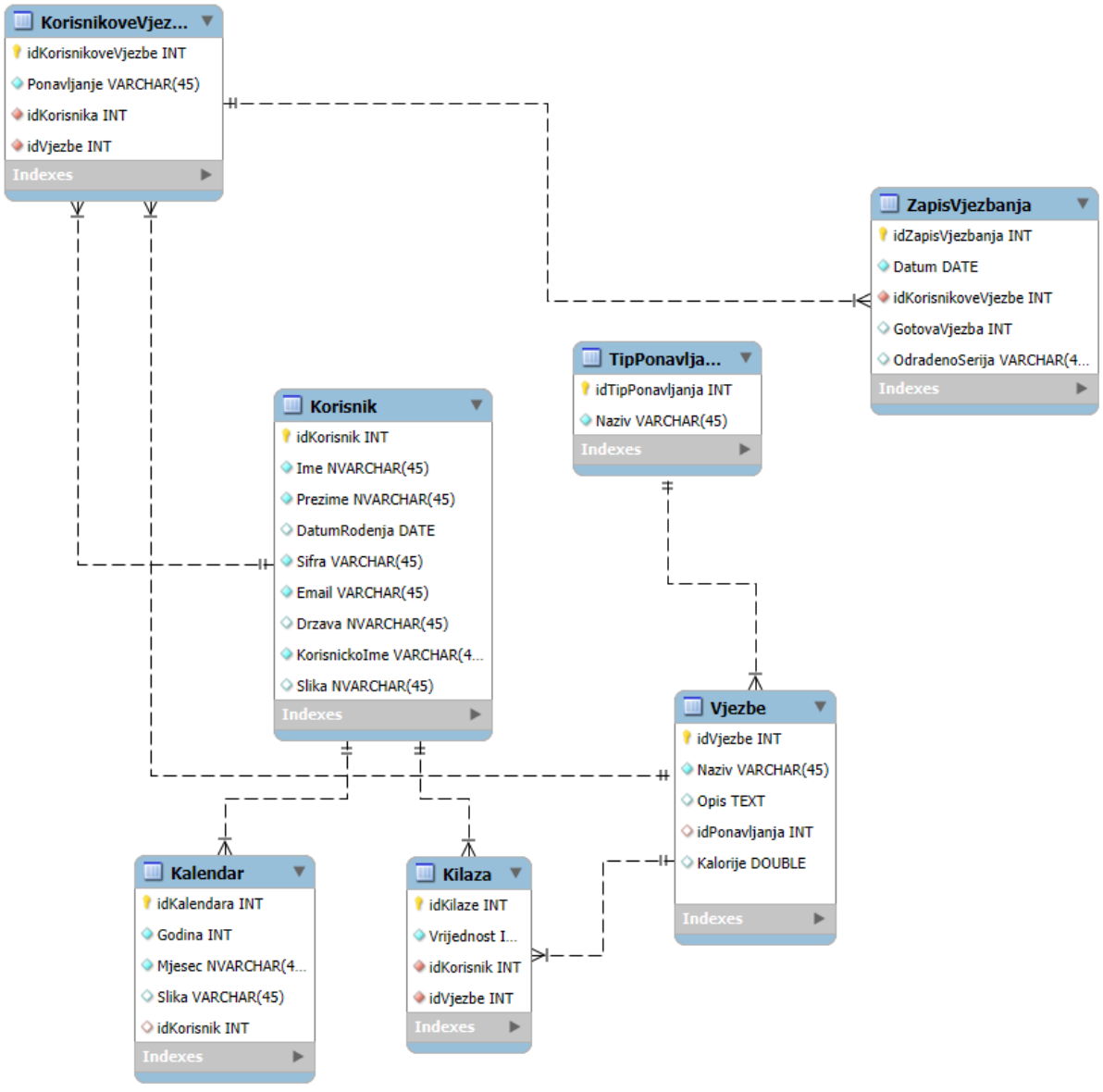
Dijagram klase prikazuje sve klase koje su prisutne u sustavu i njihove veze.



Slika 14: Dijagram klasa [vlastita izrada]

5.2.5. ERA dijagram

ERA dijagram prikazuje sve tablice u bazi podataka koje se koriste u sustavu. Dijagram prikazuje sve njihove veze zajedno sa njihovim primarnim i vanjskim ključevima.



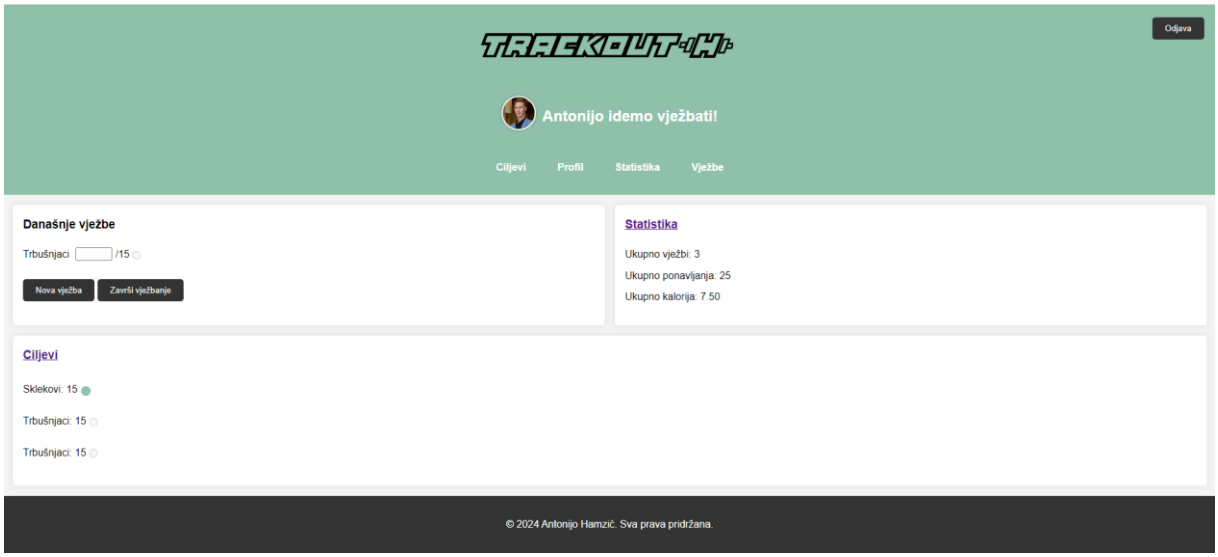
Slika 15: ERA dijagram [vlastita izrada]

5.3. Prikaz aplikacije

5.3.1. Glavna stranica

Glavna stranica sadrži većinu elemenata tako da će najveći fokus biti na njoj. Na glavnoj stranici nalazi se zaglavlje s logotipom stranice, gumbom za odjavu, profilnom slikom i navigacijom.

Glavni dio stranice u sebi sadrži tri odlomka. Svaki odlomak ima posebno definiranu stranicu sa više informacija. Prvi odlomak služi za unos i izvršavanje vježba. Drugi odlomak statistika sadrži osnovnu statistiku o današnjim vježbama. Treći odlomak ciljevi sadrži sve izvođene današnje vježbe.



Slika 16: Glavna stranica [vlastita izrada]

Dio s unosom današnjih vježba također sadrži prozor za unos podataka.

Slika 17: Prozor unosa [vlastita izrada]

HTML cijele stranice nije toliko kompliciran. Ima par elemenata i svaki mora imati svoj id kako bi se mogao kasnije koristiti u JavaScriptu. Postoji definiran gumb za novu vježbu i završavanje vježbe te kod za prozor.

```
<div>
        <h3>Današnje vježbe</h3>
        <ul id="danasnjeVjezbe"></ul>
        <button id="novaVjezbaBtn">Nova vježba</button>

        <dialog id="novaVjezbaDijalog">
          <li>
            Vježbe:
            <select id="vjezbeLista"></select>
          </li>
          <li>
            Ponavljanja:
            <input type="number" />
          </li>
          <button id="dodajVjezbu">Dodaj</button>
          <button id="zatvoriDijalog">Zatvori</button>
        </dialog>
        <button id="zavrsiVjezbanje">Završi vježbanje</button>
      </div>
```

Svaka stranica u ovoj aplikaciji kao i ova glavna ima definirane pozive funkcija koje se moraju odmah izvesti unutar slušatelja događaja za prozor. Što znači da će se sve funkcije pozvane unutar tog slušatelja događaja dogoditi odmah pri pokretanju prozora.

```
window.addEventListener("load", async () => {})
```

Kako bi prozor bio ispunjen sa potrebnim vježbama prvo se moraju dohvatiti vježbe te se moraju postaviti unutar padajuće liste. Dohvaćanje svakog podataka unutar aplikacije funkcionira na dosta sličan način. Prvo se provjerava lokalno spremište kako bi se vidjelo postojanje podataka. Ako ne postoje podaci unutar lokalnog spremišta zove se funkcija za dohvaćanje podataka iz baze podataka.

```
let Vjezbe = JSON.parse(localStorage.getItem("Vjezbe"));
if (!Vjezbe) {
  VjezbeRezultat = await dajVjezbe();
  Vjezbe = VjezbeRezultat.result;
}
```

Kroz funkciju dajVjezbe će se pokazati kako se elementi iz baze podataka dohvaćaju unutar ove aplikacije. Ako je poziv funkcije uspješan i ako su se dobili podaci oni se moraju spremati u lokalno spremište kako se pri sljedećem pokretanju stranice ne bi trebali dohvaćati izravno iz baze podataka.

```
async function dajVjezbe() {
  let parametri = { method: "GET" };
  let odgovor = await fetch("baza/vjezbe", parametri);
  if (odgovor.status == 200) {
    let podaci = await odgovor.text();
    podaci = JSON.parse(podaci);
    localStorage.setItem("Vjezbe",
JSON.stringify(podaci.result));
    return podaci;
  }
}
```

Kako bi ovo dohvaćanje radilo mora se definirati i poslužiteljski dio aplikacije. Najprije unutar već postavljenom poslužitelju moraju se postaviti metode. Metoda za dohvaćanje vježba je:

```
app.get("/baza/vjezbe", rest.getVjezbe);
```

Kao što se vidi iz metode moraju se definirati i rest funkcije. Rest funkcija za dobivanja vježba:

```
exports.getVjezbe = function (zahtjev, odgovor) {
  odgovor.type("application/json");
  odgovor.status(200);
  let dao = new DAO();
  dao.dajVjezbe().then((Vjezbe) => {
    odgovor.send(JSON.stringify(Vjezbe));
  });
};
```

Nakon te metode moraju se definirati unutar DAO.js funkcije za dohvaćanje samih podataka iz baze korištenjem SQL-a

```
dajVjezbe = async function () {
  this.baza.spojiSeNaBazu();
  let sql = "SELECT * FROM Vjezbe";
  let podaci = await this.baza.izvrsiUpit(sql, []);
  const rezultat = { result: podaci, length: podaci.length };
  this.baza.zatvoriVezu();
  return rezultat;
};
```

Na istom principu radi većina dohvaćanja, spremanja i mijenjanja podataka iz baze podataka tako da se neće prolaziti kroz to za ostale bitne funkcije ove mrežne aplikacije.

Još jedna korist lokalnog spremišta na ovoj stranici je kod unosa samih vježba. Naime kako korisnik može promijeniti unos napravljenih ponavljanja vježbe, tako se dinamički samostalno mijenja i zapis tog podataka u lokalnom spremištu. Te se kasnije i samostalno računa i statistika bez ponovnog ulaska u bazu podataka. Kod svakog pokreta aplikacije i dodavanje novih vježba pokrenuta je funkcija ispisiListu. Bitan dio koda te funkcije je dio koji ispisuje HTML i koji u sebi sadrži poziv funkcije provjeriVrijednost koja se pokreće pri svakoj promjeni broja.

```
vjezbe.forEach((vjezba) => {
    const li = document.createElement("li");
    if (vjezba.GotovaVjezba == 0) {
      li.innerHTML = `
            ${vjezba.Naziv}
                  <input type="number" name="${vjezba.Naziv}"
style="width: 50px;" min="0" max="${vjezba.Ponavljanje}"
oninput="provjeriVrijednost(this,${vjezba.Ponavljanje},${vjezba.idZa
pisVjezbanja})" > /${vjezba.Ponavljanje}
                              <input type="radio" disabled
name="${vjezba.Naziv}+${vjezba.idZapisVjezbanja}">
          `;

      listaVjezbi.appendChild(li);
    }
  });
```

Funkcija provjeriVrijednost provjerava vrijednost unosa te ovisno o uspješnosti označuje gumb. Također sprema u lokalno spremište broj odrađenih ponavljanja uz id vježbe.

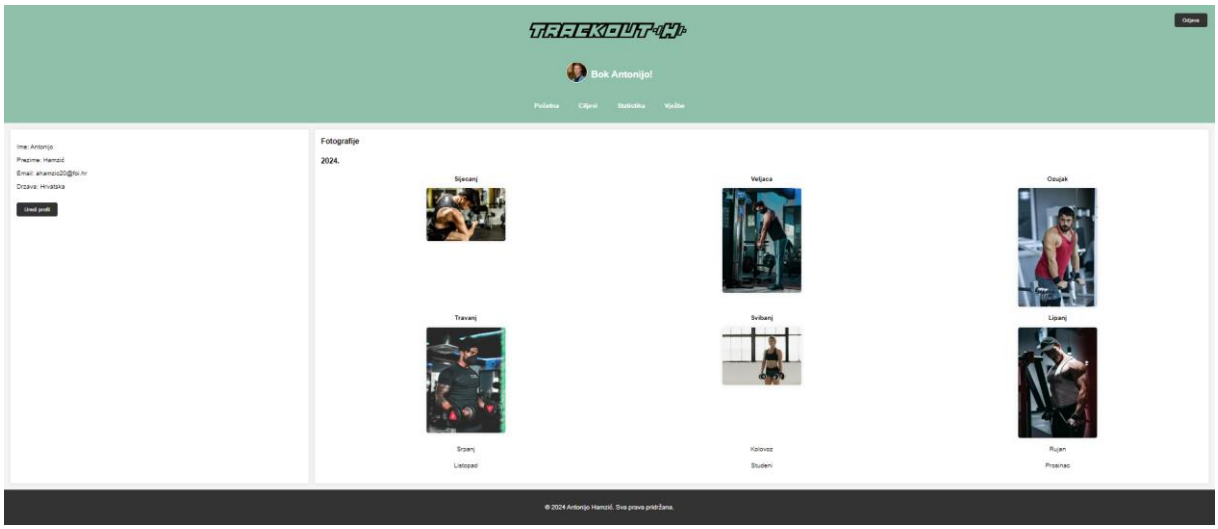
```
function provjeriVrijednost(unos, ponavljanja, id) {
  const radioButton = unos.nextElementSibling;

  if (unos.value == ponavljanja) {
    radioButton.checked = true;
  } else {
    radioButton.checked = false;
  }

  localStorage.setItem(id, unos.value);
}
```

Ta vrijednost je bitna kada korisnik pritisne gumb za odrađenu vježbu te program pozove funkciju napraviVjezbe. Funkcija napraviVjezbe sprema podatke o vježbama u tablicu zapisa svih vježba koje su odrađene ili nisu odrađene. Prikazat će se samo isječak funkcije koja priprema podatke za slanje.

```
let podaci = {
        idZapisVjezbanja: vjezba.idZapisVjezbanja,
        serije: localStorage.getItem(vjezba.idZapisVjezbanja),
      };
```

5.3.2. Profil

Stranica profila ima tri funkcionalnosti unutar sebe. Na profilnoj stranici se može promijeniti slika profila, promijeniti podatke o korisniku, pregledavati i promijeniti kalendar fotografija korisnika



Slika 18: Profil stranica [vlastita izrada]

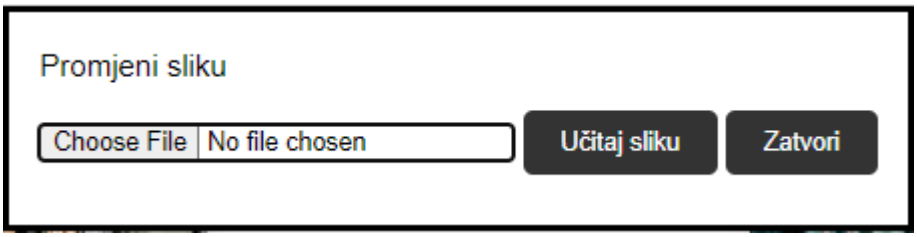
Specifičnost ove stranice je učitavanje i prikazivanje slika. Koristi se kod galerije fotografija i kod profilne slike. Postupak će se prikazati preko postavljanja slika iz galerije. Postupak mijenjanja ili postavljanja slike u kalendar započinje pritiskom na tekst mjeseca. HTML galerije sadrži prozore za unos slike i popis svih mjeseci.

```
<div>
        <h3>Fotografije</h3>
        <h3>2024.</h3>
        <ul class="fotografije">
          <li id="sijecanj">Siječanj</li>
          <li id="veljaca">Veljača</li>
          <li id="ozujak">Ožujak</li>
          <li id="travanj">Travanj</li>
          <li id="svibanj">Svibanj</li>
          <li id="lipanj">Lipanj</li>
          <li id="srpanj">Srpanj</li>
          <li id="kolovoz">Kolovoz</li>
          <li id="rujan">Rujan</li>
          <li id="listopad">Listopad</li>
          <li id="studeni">Studeni</li>
          <li id="prosinac">Prosinac</li>
        </ul>
        <dialog id="urediSlikuDijalog">
          Promjeni sliku
          <br />
          <input
            type="file"
            id="imageUploadSlike"
            name="image"
            accept="image/*"
            required
          />
          <button id="promijeniSliku">Upload Image</button>
          <button id="zatvoriDijalogSlike">Zatvori</button>
          <div id="uploadStatus"></div>
        </dialog>
      </div>
```

Pritiskom na mjesec se poziva slušatelj događaja koji mijenja vrijednost varijable ovisno o pritisnutom mjesecu i poziva otvorenje prozora.

```
document.querySelectorAll(".fotografije li").forEach((li) => {
    li.addEventListener("click", () => {
      mjesec = li.id;
      urediSlikuDijalog.showModal();
    });
  });
```

Prikaz otvorenog prozora:



Slika 19: Prozor za dodavanja slike [vlastita izrada]

Kada se slika učita poziva se slušatelj događaja koji prvo zove funkciju uploadajSliku koja sprema sliku i ako je sve uredu funkciju spremiSliku.

```
promjeniSliku.addEventListener("click", async () => {
    const fileInput = document.getElementById("imageUploadSlike");
    const file = fileInput.files[0];
    let podaci = await uploadajSliku(file);
    if (podaci) {
       podaci.Mjesec = mjesec;
       await spremiSliku(podaci);
     }
});
```

Funkcija uploadajSliku sprema sliku u mapu upload te sprema naziv datoteke u bazu podataka. Spremanje u bazu podataka je slično kao dohvaćanje vježba iz baze podataka te neće biti prikazano ovdje.

```
async function uploadajSliku(file){
      const formData = new FormData();
      formData.append("image", file);
      const odgovor = await fetch("/upload", {
            method: "POST",
            body: formData,
      });
  const rezultat = await odgovor.json();
}
```

Da bi to sve radilo unutar poslužitelja je potrebno dodati način spremanja i samu funkciju koja se zove.

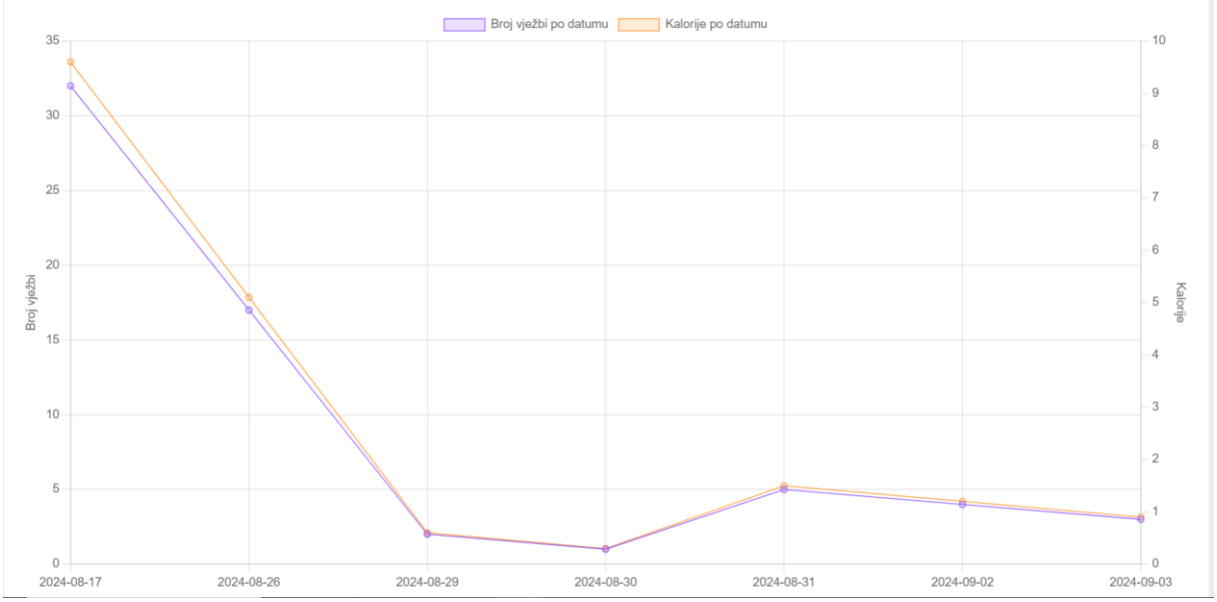
```
      const storage = multer.diskStorage({
        destination: (req, file, cb) => {
          cb(null, "uploads/");
        },
        filename: (req, file, cb) => {
          cb(null,                  Date.now()                  +
path.extname(file.originalname));
        },
      });

      const upload = multer({ storage: storage });
      app.post("/upload", upload.single("image"), (req, res) =>
{
  if (!req.file) {
    return res.status(400).json({ message: "No file uploaded."
});
  }
  res
    .status(200).json({
     message: "Image uploaded successfully!",
     filename: req.file.filename,
    });
});
```

Na kraju je potrebno prikazati tu sliku. Zato se poziva funkcija promjeniSlikuKalendara unutar funkcije koja sprema fotografiju u bazu.

```
async function promjeniSlikuKalendara(podaci) {
  let slika = document.getElementById(podaci.Mjesec);
  slika.innerHTML = `
  <span>${slika.id.charAt(0).toUpperCase()                    +
slika.id.slice(1)}</span>
    <a href="uploads/${podaci.Slika}" target="_blank">
      <img src="uploads/${podaci.Slika}" alt="Slika" width=200>
    </a>
`;
  const imageLink = slika.querySelector("a");
  imageLink.addEventListener("click", (event) => {
    event.stopPropagation();
  });
}
```

5.3.3. Statistika

Stranica statistike uz prikaz računa statistike također sadrži grafove. Jedan od tih grafova prikazuje broj vježbi po datumu i broj kalorija potrošenih po datumu.



Slika 20: Prikaz grafa iz stranice [vlastita izrada]

Kako bi bio moguć rad s grafovima u HTML-u stranice mora se dodati posebna skripta.

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

Također se mora dodati i element canvas preko kojeg se unosi graf.

```
<canvas id="grafVjezbi" width="400" height="200"></canvas>
```

Zatim se u JavaScriptu mora definirati funkcija za kreiranje grafa. U ovom slučaju to je funkcija kreirajGraf. Funkcija kreirajGraf prima podatke o odrađenim vježbama. Kada se dobiju potrebni podaci, mora se graf nacrtati. Prije svakog crtanja potrebno je uništiti prošlu instancu grafa. Unutar izrade grafa se moraju ispuniti potrebne informacija (oznake, podaci, boje linija). Prikazat će se samo izrada lijeve strane grafa (broj vježbi) zbog prevelike količina koda.

```
let istancaGrafa;

function kreirajGraf(vjezbe) {
const ctx = document.getElementById("grafVjezbi").getContext("2d");
        const brojDatuma = {};
```

```
vjezbe.forEach((vjezba) => {
  const date = vjezba.Datum;
  brojDatuma[date] = (brojDatuma[date] || 0) + 1;
});

const labele = Object.keys(brojDatuma);
const podatciVjezbe = Object.values(brojDatuma);

if (istancaGrafa) {
  istancaGrafa.destroy();
}

istancaGrafa = new Chart(ctx, {
  type: "line",
  data: {
    labels: labele,
    datasets: [
      {
        label: "Broj vježbi po datumu",
        data: podatciVjezbe,
        backgroundColor: "rgba(153, 102, 255, 0.2)",
        borderColor: "rgba(153, 102, 255, 1)",
        borderWidth: 1,
        yAxisID: "y1",
      }
    ],
  },
```

```
    options: {
      scales: {
        y1: {
          type: "linear",
          position: "left",
          beginAtZero: true,
          title: {
            display: true,
            text: "Broj vježbi",
          },
        },
      },
    },
  });
}
```

6. Zaključak

U ovom završnom radu se istražila tema lokalnih spremišta u HTML5. Kroz teorijski dio razradile su se tehnologije koje omogućuju pohranu podataka na klijentskoj strani. Prvo su se prikazale osnovna svojstva JavaScripta u kojem se nalazi lokalno spremište. Prošlo se kroz svojstva, prednosti i nedostataka lokalnog načina spremanja podataka. Uz sve vrste lokalnih prolazilo se i kroz ostale vrste spremišta kao što su udaljeno i hibridno. Također se prolazilo kroz sinkronizacije lokalnih i udaljenih spremišta podataka.

Za praktični dio završnog rada zadatak je bio izraditi mrežnu aplikaciju koja se koristi lokalnim spremištima. Uz samu izradu aplikacije izradili su se i određeni dijagrami koji su pomogli kod razvoja aplikacije.

Kroz razradu teorijskog i praktičnog djela rada, moglo se zaključiti da su lokalna spremišta bitna u izradi modernih mrežnih aplikacija. Dosta je korisno da aplikacija nije ovisna o mreži korisnika te da može raditi i bez internetske veze. Unatoč određenim ograničenjima, kao što su ograničeni kapaciteti i sigurnost, lokalna spremišta pružaju veće prednosti u kontekstu performansa i korisničkog iskustva. Omogućuju brži dohvat podataka u usporedbi sa udaljenim spremištima. Naravno za dobro korištenje lokalnog spremišta programer mora biti upućen u sve prednosti i nedostatke takvog rada. Tako da na pogled jednostavni način spremanja podataka može imati svoje zamke i izazove. Programer mora pažljivo razmotriti kontekst u kojem koristi lokalno spremište te implementirati odgovarajuće mjere kako bi osigurao optimalno i sigurno korištenje.

Popis literature

[1] „HTML Web Storage API“. [Na internetu]. Dostupno na: https://www.w3schools.com/html/html5_webstorage.asp. [Pristupljeno: 31-svi-2024.].

[2] "HTML5 Storage". [Na internetu]. Dostupno na: https://www.gwtproject.org/doc/latest/DevGuideHtml5Storage.html. [Pristupljeno: 31-svi-2024.].

[3] "What are the different types of storage in HTML5?". [Na internetu]. Dostupno na: https://www.geeksforgeeks.org/what-are-the-different-types-of-storage-in-html5/. [Pristupljeno: 31-svi-2024.].

[4] "JavaScript Web API". [Na internetu]. Dostupno na: https://www.w3schools.com/jsref/api_web.asp. [Pristupljeno: 31-svi-2024.].

[5] "Local Storage vs Session Storage vs Cookie". [Na internetu]. Dostupno na: https://www.xenonstack.com/insights/local-vs-session-storage-vs-cookie. [Pristupljeno: 31-svi-2024.].

[6] "The Different Types of Storage in HTML 5". [Na internetu]. Dostupno na: https://www.c-sharpcorner.com/blogs/the-different-types-of-storage-in-html-5. [Pristupljeno: 31-svi-2024.].

[7] Flanagan, D. (2020). JavaScript: The Definitive Guide (7th ed.). O'Reilly Media, Inc.

[8] Verou, L., Zhang, A. X., & Karger, D. R. (2016). Mavo: Creating interactive data-driven web applications by authoring HTML. Proceedings of the 29th Annual Symposium on User Interface Software and Technology.

[9] Bogaard, D., Johnson, D., & Parody, R. (2012). Browser web storage vulnerability investigation: HTML5 localstorage object. Proceedings of the International Conference on Security and Management.

[10] Parra Rodriguez, J. D., & Posegga, J. (2018). Local storage on steroids: Abusing web browsers for hidden content storage and distribution. In J. Lopez, J. Zhou, M. Theoharidou, S. Marsh, & J. Garcia-Alfaro (Eds.), Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part II (pp. 275-293). Springer International Publishing.

[11] Dalimunthe, S., Reza, J., & Marzuki, A. (2022). The model for storing tokens in local storage (Cookies) using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in e-learning systems. Journal of Applied Engineering and Technological Science (JAETS), 3(2), 149-155.

[12] iCode Academy. (2017). HTML5 & CSS3 for beginners: Your guide to easily learn HTML5 & CSS3 programming in 7 days.

[13] Duckett, J. (2014). JavaScript and JQuery: Interactive Front-End Web Development. Wiley.

[14] "What is the Document Object Model? DOM for Beginners". [Na internetu]. Dostupno na : https://www.freecodecamp.org/news/introduction-to-the-dom/. [Pristupljeno: 17-kol-2024.]

[15] "CouchDB sync". [Na internetu]. Dostupno na : https://pouchdb.com/guides/replication.html#couchdb-sync. [Pristupljeno: 20-kol-2024.]

Popis slika