Lokalno spremište podataka u HTML5

Sokolić, Josip

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Sokolić

LOKALNO SPREMIŠTE PODATAKA U HTML5

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Sokolić

Matični broj: S43492/14-I

Studij: Primjena informacijske tehnologije u poslovanju

LOKALNO SPREMIŠTE PODATAKA U HTML5

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, listopad 2018.

Josip Sokolić

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

Kao što naslov rada upućuje tematika koja se obrađuje su lokalna spremišta podataka u HTML5. No za početak se obrađuje HTML radi lakšeg razumijevanja sadržaja koji slijedi, ali i da se dobije perspektiva kako je uopće došlo od jednostavnog prezentacijskog jezika do lokalnih spremišta podataka. Nakon što se utvrde osnove HTML-a prelazi se na prvo spremište podataka, a to je lokalno spremište podataka. U tom poglavlju su pokrivene sve osnove koje su potrebne za korištenje lokalnog spremišta podataka pri izradi mrežne aplikacije, te su potkrepljene primjerima iz prakse. Povučene su paralele sa nekim drugim načinima koji su korišteni prije pojave lokalnog spremišta podataka. Nakon toga se nastavlja istim tempom na iduće spremište podataka – sesijsko spremište podataka. Ovdje je priča vrlo slična lokalnom spremištu podataka uz male razlike koje su objašnjene, te također potkrijepljene mnoštvom primjera. Za kraj teorijskog dijela rada se prelazi na zadnje lokalno spremište podataka, a to su indeksirane baze podataka. Ovo poglavlje je takoreći naprednije, te su koncepti osebujniji nego što je bio slučaj sa prethodna dva spremišta podataka. Ovdje će se upoznati i demonstrirati baratanje sa programskim sučeljima i mnogo više. Nakon završetka teorijskih poglavlja, za kraj preostaje upoznavanje sa aplikacijom “eGlas“, te demonstracijom načela iz teorijskog dijela na konkretnim primjerima iz same aplikacije.

Sadržaj

Sadržaj iii
1. Uvod 1
2. Metode i tehnike rada 2
2.1. Javascript 2
2.2. CSS 2
2.3. jQuery 2
2.4. Ajax 2
2.5. SQL 3
2.6. Bootstrap 3
2.7. PHP 3
3. HTML 4
3.1. Što je HTML 4
3.2. Povijest HTML jezika 5
3.2.1. HTML oznake 5
3.2.2. Verzija 2.0 5
3.2.3. Verzija 3.0 6
3.2.4. HTML4 6
3.2.5. HTML5 7
3.3. Struktura HTML dokumenta 7
3.3.1. Građevni blokovi 7
3.3.2. Struktura HTML elementa 8
3.3.3. Primjer HTML dokumenta 9
3.4. Komentari 10
3.4.1. Struktura komentara 10
3.4.2. Meta elementi 11
3.5. Dodavanje zvuka, slike i multimedije 11
3.5.1. Dodaci mrežnog preglednika 12
3.5.2. ActiveX kontrole 12
3.5.3. Java umetak 12
3.5.4. Zvuk i multimedija 13
3.5.5. Slike 14
4. Lokalno spremište 16
4.1. Što je lokalno spremište 16
4.1.1. Struktura i pristupačnost 17

4.2. Razlika između lokalnog spremišta i kolačića 17
4.2.1. Kolačići 18
4.2.2. Usporedba 19
4.3. Sigurnost lokalnog spremišta 20
4.3.1. Napadi podvale sustava domene 20
4.3.2. Križni napadi direktorija 21
4.3.3. Implementacijski rizici 21
4.4. Događaj lokalnog spremišta 21
4.4.1. Događaj 22
4.4.2. Slušatelji događaja 22
4.4.3. Lokalni događaji 23
4.5. Uporaba lokalnog spremišta 24
4.5.1. Funkcije objekta lokalnog spremišta podataka 24
4.5.2. Primjer iz prakse – sistemski gumb "povratak" 26
5. Sesijsko spremište 30
5.1. Što je sesijsko spremište 30
5.2. Sigurnost sesijskog spremišta 31
5.2.1. Napadi podvale sustava domene 31
5.2.2. Križni napadi direktorija 32
5.2.3. Implementacijski rizici 32
5.2.4. Razlika između sesijskog spremišta i lokalnog spremišta sa aspekta sigurnosti 32
5.3. Događaj sesijskog spremišta 33
5.4. Uporaba sesijskog spremišta 34
5.4.1. Funkcije objekta sesijskog spremišta podataka 34
5.4.2. Primjer iz prakse - prijava 36
6. Indeksirana baza podataka 39
6.1. Što je indeksirana baza podataka 39
6.1.1. Osnovni koncepti indeksirane baze podataka 40
6.1.2. Programska sučelja indeksiranih baza podataka 41
6.2. Sigurnost indeksirane baze podataka 42
6.2.1. Dijeljenje resursa između različitih izvorišta 42
6.2.2. Skriptiranje sa udaljenih stranica 42
6.2.3. Napadi društvenog inženjerstva 43
6.2.4. Fizički pristup 43
6.3. Događaji indeksirane baze podataka 43
6.3.1. Događaj upgradeneeded 43
6.3.2. Događaj complete 45

6.3.3. Događaj abort 46
6.3.4. Događaj success 47
6.3.5. Događaj error 48
6.3.6. Događaj blocked 49
6.3.7. Događaj versionchange 50
6.3.8. Događaj close 51
7. Aplikacija eGlas 52
7.1. Općenito o aplikaciji 52
7.2. Korisničko sučelje 53
7.3. Programski kod 59
8. Zaključak 67
Popis literature 69
Popis slika 70
Popis tablica 71

1. Uvod

Cilj ovog rada je kroz razne tehnologije i primjer aplikacije prikazati glavne principe i metodike rada na primjerima mrežnih aplikacija i stranica. Dakako glavni fokus će biti na zadnjoj verziji HTML specifikacije, a to je HTML5. Nešto više o HTML-u će se saznati iz poglavlja posvećenom HTML-u, njegovoj povijesti, motivaciji za njegov napredak i kakvo mjesto danas uživa među mrežnim tehnologijama. No neće se ići u punu širinu HTML5 specifikacije, glavni orijentir u radu će biti lokalna spremišta podataka, te kako se ona uklapaju u današnje mrežno programiranje. Pokazat će se na velikom broju primjera kako lokalna spremišta podataka mogu biti glavni saveznik jednom mrežnom programeru. S obzirom da je dio ovog rada i aplikacija u kojoj se koriste svi principi koji su obrađeni u radu, svrha teorijskog dijela je da postane očigledno gdje se u aplikaciji koristi lokalno spremište podataka i zašto se koristi. Također je bitno imati na umu da postoje alternative za lokalna spremišta podataka, ali isto tako je bitno znati da se nazivaju alternativama s razlogom. Razlog je taj što velika većina mrežnih programera upravo bira lokalna spremišta podataka u radu sa mrežnim tehnologijama. Pogotovo u današnje vrijeme gdje korisnička računala postaju sve jača i jača, a poslužitelji teško prate korak. Što zapravo znači da nedostatak resursa na klijentskoj strani više nije slučaj, u novije vrijeme se zapravo počeo stvarati suficit resursa koji je vrlo praktičan za korištenje za realizaciju nekih funkcionalnosti mrežnih aplikacija i zbog manje potrebe za komunikaciju sa poslužiteljem poboljšava performanse tih istih mrežnih aplikacije. Naravno nije problem u nedostatku resursa na poslužitelju nego u financijskom aspektu zakupljivanja većeg broja resursa. Lokalna spremišta podataka spremaju podatke i barataju njima direktno na korisničkom računalu, te na taj način pružaju uštedu poduzećima, što sa financijskog aspekta, a što sa aspekta uštede resursa. U ovom radu je zapravo glavna motivacija pokazati spomenute benefite lokalnih spremišta podataka.

2. Metode i tehnike rada

U ovom poglavlju biti će objašnjene sve tehnologije koje će biti korištene kroz rad i aplikaciju. Bilo za primjere ili kao pomoćni alati za objašnjavanje osnovnih načela suštine ovog rada, a to su lokalna spremišta podataka i baratanje njima.

2.1. Javascript

Javascript je skriptni jezik koji omogućuje implementaciju složenijih stvari prilikom izrade mrežnih stranica. Svaki dio mrežne stranice koji nije statičan je djelo javascript skriptnog jezika. On se još naziva i trećim slojem skupine standardinh mrežnih tehnologija koje uz njega čine još i CSS i HTML.[1], [2]

2.2. CSS

CSS (*eng. Cascading Style Sheets)* je stilski jezik koji se koristi za opisivanje prezentacije dokumenta napisanog u HTML-u. Kao što je ranije spomenuto CSS je također jedna od tri temeljne mrežne tehnologije. CSS je dizajniran kako bi omogućio odvajanje prezentacije od sadržaja, uključujući izgled, boje i fontove. [3]

2.3. jQuery

JQuery je programski okvir (*eng. framework)* za javascript. Svrha jQuery-a je olakšati korištenje javascript jezika prilikom izrade mrežnih stranica. Jquery odrađuje mnoge uobičajene zadatke, za koje bi inače trebalo mnogo više linija javascript koda, te ih pakira u metode koje se mogu pozvati u jednoj liniji koda. Također pojednostavljuje mnoge komplicirane stvari iz javascript jezika poput ajax poziva.[4]

2.4. Ajax

Ajax (*eng. Asynchronus Javascript and XML)* je skup tehnika za mrežni razvoj aplikacija koji koriste mnoge mrežne tehnologije na strani klijenta za stvaranje asinkronih mrežnih aplikacija. Pomoću ajax poziva mrežne aplikacije mogu slati i preuzimati podatke s poslužitelja asinkrono, tj. u pozadini bez ometanja prikaza i ponašanja postojeće stranice. Odvajanjem slojnog razmaka podatka iz prezentacijskog sloja, ajax dopušta dinamičku izmjenu sadržaja mrežne stranice bez potrebe za ponovni učitavanjem strance.[5]

2.5. SQL

SQL (*eng. Structured query language)* je jezik koji se koristi u programiranju i namijenjen je za upravljanje podacima koji se održavaju u relacijskom sustavu za upravljanje bazama podataka ili za obradu tokova u relacijskom toku podataka. To je osobito korisno u upravljanju strukturiranim podacima gdje postoje odnosi između različitih entiteta.[6]

2.6. Bootstrap

Bootstrap je besplatan okvir za brže i jednostavnije razvijanje prednjeg kraja (*eng. front-end)* mrežnih aplikacija i stranica. Bootstrap uključuje HTML i CSS za temeljne dizajnerske predloške za tipografiju, obrasce, gumbe, tablice, navigaciju, modale, slike vrtuljaka i mnoge druge elemente. Također uz njega ide i javascript dio koji uključuje posebnu kontrolu za svaki element koji je definiran unutar Bootstrap okvira, osim ako je element statičan. No najbitnija značajka planetarne popularnosti Bootstrap okvira je jednostavnost prilikom implementacije osjetljivog dizajna (*eng. responsive design)*.

2.7. PHP

PHP je skriptni jezik koji se odrađuje na strani poslužitelja, te je dizajniran za razvoj mrežnih stranica, ali također se koristi kao programski jezik opće namjene. Izvorno je kreiran od strane Rasmusa Lerdorfa 1994. godine. Trenutnu referentnu implementaciju proizvodi i održava The PHP Group. PHP je izvorno značio osobna početna stranica (*eng. Personal home page)*, no sada se radi o rekurzivnom akronimu – hipertekstualni predprocesor (*eng. Hypertext Preprocessor*).[7]

3. HTML

U ovom poglavlju naslovljenom HTML obrađuje se jedan od najbitnijih aspekata cijelog rada, a to je kako i naslov navodi HTML jezik. Za početak se kreće sa nečim općenitim o samom HTML jeziku, a na to se nadovezuje povijest HTML jezika, kako je nastao, kada, koja je bila glavna motivacija, te zašto i kako se javila potreba za takvim rješenjem. Nakon toga nastavlja se sa nečim malo više vezanim za samu praktičnu primjernu HTML jezika, a to će za početak biti sama struktura HTML jezika koja je ključna za razumijevanje i uporabu HTML jezika u praksi. Kada se utvrde nužne osnove za korištenje HTML jezika prelazi se na komentare u HTML jeziku. Komentari su važan dio HTML jezika, jer pomoću njih se ostvaruje organizacija, te olakšava čitljivost napisanog koda (*eng. code*). Za kraj ovog poglavlja biti će objašnjeno, te pokazano kroz nekolicinu praktičnih primjera kako koristiti HTML jezik za dodavanje tj. prikaz zvuka, slike i multimedijskih sadržaja.

3.1. Što je HTML

HTML je kratica za Hypertext Markup Language, što zapravo znači, prezentacijski jezik za izradu web stranica. Pomoću HTML jezika stvaramo tako zvani hipertekst (*eng. hypertext*) dokument. Uz pomoć HTML jezika oblikuje se sadržaj i stvaraju se hiperveze (*eng. hyperlink*) hipertekst dokumenta. HTML je jednostavan za uporabu i lako se uči, što je jedan od razloga njegove opće prihvaćenosti i popularnosti. Svoju raširenost zahvaljuje jednostavnosti i tome što je od početka bio zamišljen kao besplatan i tako dostupan svima. Prikaz hipertekst dokumenta omogućuje mrežni preglednik (*eng. web browser*). Temeljna zadaća HTML jezika jest uputiti mrežni preglednik kako prikazati hipertekst dokument. Pri tome se nastoji da taj dokument izgleda jednako bez obzira o kojemu je mrežnom pregledniku, računalu i operacijskom sustavu riječ. Važno je napomenuti da HTML jezik nije programski jezik niti su ljudi koji koriste isključivo HTML jezik programeri, no u današnje vrijeme gotovo više ne postoje ljudi koji koriste isključivo i samo HTML jezik. Uz HTML jezik se najčešće koristi CSS (*eng. Cascading Style Sheets*) i Javascript, jer je poznavanje te tri tehnologije nužno za radno mjesto razvojnog programera korisničke strane (*eng. front-end developer*). HTML jezikom se ne može izvršiti nikakva zadaća, pa čak ni najjednostavnija operacija zbrajanja ili oduzimanja dvaju cijelih brojeva. On služi, kako je i vidljivo iz samog naziva, samo za opis naših hipertekstualnih (*eng. hypertext*) dokumenata. HTML datoteke su zapravo po svojoj strukturi samo obične tekstualne datoteke, ekstenzija im je .html ili .htm.

Osnovni građevni elementi svake stranice su oznake (*eng. tag*) koji opisuju kako će se nešto prikazati u mrežnom pregledniku. Poveznice unutar HTML dokumenata povezuju

dokumente u uređenu hijerarhijsku strukturu i time određuju način na koji posjetitelj doživljava sadržaj stranica.

3.2. Povijest HTML jezika

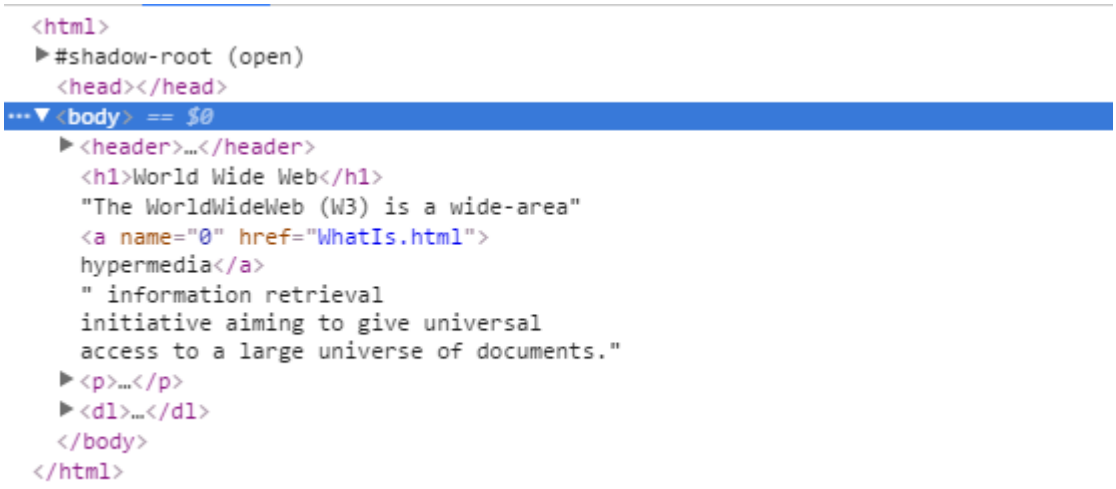
Nakon što je objašnjeno što je zapravo HTML jezik, zašto se koristi, te koje su njegove mogućnosti i ograničenja, fokus će biti prebačen na povijest HTML jezika kako bih se pokazao razvoj, te glavna motivacija koja stoji iza tog razvoja HTML jezika.

3.2.1. HTML oznake

Prvi javno dostupan opis HTML jezika je dokument zvan “HTML tags“, prvi put se spominje na internetu od strane Tim Berners-Leeja krajem 1991. godine. Taj opis se sastoji od 20 elemenata početnog, relativno jednostavnog dizajna HTML jezika. Trinaest od tih inicijalnih 20 elemenata još uvijek postoji u trenutnoj verziji HTML jezika koja se zove HTML5. HTML jezik nastanak mnogih svojih oznaka (*eng. tag*) duguje jednom od ranijih jezika za formatiranje teksta, Runoff jeziku. Runoff jezik je razvijen u ranim 1960-im godinama za operacijski sustav koji se zvao Compatible Time-Sharing System ili skraćeno CTSS. Runoff je kasnije inkorporiran u UNIX operativni sustav kao dio naprednijih formatirajućih programa kao što su roff, nroff i troff. Svaka nova verzija HTML jezika je razvijana tako da ostane čitljiva na svim mrežnim preglednicima. Tim Berners-Lee je, nakon što je u listopadu 1994. napustio Europsku organizaciju za nuklearno istraživanje koja se zove CERN, osnovao organizaciju koju je nazvao World Wide Web Consortium koja se bavi standardizacijom tehnologija korištenih na webu poznatija kao W3C.

3.2.2. Verzija 2.0

Prva verzija HTML jezika objavljena je 1993. godine. U to je vrijeme HTML jezik bio još poprilično ograničen. Jedna od mogućnosti koje tada još nisu bile inkorporirane u njega je bilo dodavanje slika u HTML dokumente. Razvoj HTML jezika nastavljen je prvom "imenovanom" verzijom – 2.0, no ni ona nije postala standardom.[8]



Slika 1: HTML kod prvog mrežnog mjesta (*eng. website*) (Izvor: http://info.cern.ch)

3.2.3.Verzija 3.0

U ožujku 1995. W3C objavljuje verziju HTML jezika imenovanom - 3.0, koja donosi mogućnosti definiranja tablica što je tada bilo velik iskorak za HTML jezik. Daljnji razvoj ove verzije HTML jezika označilo je prihvaćanje "specifičnih" oznaka podržanih u tada najvećim i najprihvaćenijim mrežnim preglednicima. Tako su nastali mnogi duplikati oznaka (*eng. tag*), tj. postojalo je više oznaka koje su imale istu funkciju.[8] Primjerice podebljani tekst (*eng. bold*) bilo je moguće definirati sa dvije različite oznake:

```
<b>
```

ali isto tako i oznakom:

```
<strong>
```

3.2.4.HTML4

Verzija HTML jezika nazvana HTML4 predstavljena je u prosincu 1997. godine. Specifikaciju spomenute verzije HTML jezika HTML4 i dalje možemo pronaći na stranicama W3C: https://www.w3.org/TR/html401/. U ovoj verziji HTML jezika i dalje se nastavilo sa prihvaćanjem oznaka nametnutih od strane proizvođača različitih mrežnih preglednika, no isto tako je pokrenuto i "čišćenje" standarda, te proglašavanje nekih od postojećih oznaka suvišnima. Manje promjene u specifikaciji ovog standarda predstavljene su u prosincu 1999. godine, kada je predstavljena i konačna verzija ovog jezika nazvana HTML4.01. koja je ostala važeća sve do prve iduće revizije standarda HTML jezika koja je i danas važeća, a nazvana je HTML5.[8]

3.2.5. HTML5

HTML5 je prva nova revizija standarda od prosinca 1999. godine, tj. od verzije HTML 4.01. Nastao je u suradnji sa World Wide Web Consortium ili skraćeno W3C i Web Hypertext Application Technology Working Group ili skraćeno WHATWG. Do 2006. godine ove su dvije grupe radile odvojeno, WHATWG je radio sa mrežnim formama (*eng. web forms*) i aplikacijama, a W3C sa XHTML 2.0. Na svu sreću odlučili su udružiti snage i kreirati novu verziju HTML jezika. Izdavanje konačnih specifikacija standarda HTML5 u suprotnosti je s inicijativom WHATWG prema kojoj bi HTML trebao biti "živi" standard koji se stalno nadograđuje, bez oznake verzije specifikacija. HTML5 donosi brojne nove mogućnosti koje prethodne verzije HTML 4.01 i XHTML 1.x nisu imali, kao što je mogućnost reprodukcije videa na stranicama bez korištenja Adobe flash ili Microsoft silverlight sustava, mogućnost upravljanja pomoću tipkovnice i opcijama za bilo koju vrstu manipulacija, vuci i pusti (*eng. drag and drop*), platno (*eng. Canvas*) kao i nekolicinom ostalih novih oznaka. Znatno više pažnje se posvećuje alternativama uporabi kolačića (*eng. Cookies*) iz čega su načelno i proizašla lokalna spremišta podataka.[8]

3.3. Struktura HTML dokumenta

U ovom poglavlju fokus se prebacuje na praktičnu primjenu HTML jezika. Kao bitan preduvjet važno je prvo upoznati strukturu samog HTML dokumenta. To će u nastavku biti objašnjeno, te pokazano na praktičnim primjerima.

3.3.1. Građevni blokovi

Svaki HTML dokument sastoji se od osnovnih građevnih blokova - HTML elemenata. Svaki, pak, HTML element sastoji se od para HTML oznaka (engl. tag). Također, svaki element može imati i atribute kojim se definiraju svojstva tog elementa.[9] Na samom početku HTML dokumenta preporučljivo je postaviti:

```
<!DOCTYPE>
```

element, kojim se označava deklariranje tipa dokumenta ili skraćeno DTD *(eng. Document Type Declaration*), čime se definira točna inačica standarda koja se koristi za izradu HTML dokumenta. Nakon DTD elementa, označava se početak HTML dokumenta sljedećom oznakom:

```
<html>
```

Unutar spomenutog elementa nalaze se još neke oznake, kao što su:

```
<head>
```

On nam predstavlja zaglavlje HTML dokumenta u kojem se najčešće specificiraju jezične značajke HTML dokumenta, te sam naslov (*eng. title*) stranice. Pomoću određenih HTML elemenata unutar zaglavlja (*eng. header*) dodaju se i stilska obilježja stanice. Ona mogu biti direktno ugrađena (*eng. embedded*) ili dodana kao referenca na vanjsku CSS (*eng. Cascading Style Sheets*) datoteku. Često se unutar zaglavlja još definiraju i skripte kreirane u JavaScript jeziku. One također mogu biti direktno ugrađene ili dodane kao referenca na vanjsku javascript datoteku. Ako se radi o referenci na vanjsku javascript datoteku, nije dobra praksa (*eng. good practice*) dodati ih unutar zaglavlja jer želimo biti sigurni da se HTML stranica cijela učitala prije izvršavanja dodatnih skripti stoga to treba izvršiti u zaglavlju HTML dokumenta (*eng. footer*). Nakon elementa zaglavlja se najčešće nalazi sljedeći element:

```
<body>
```

Što se samog sadržaja stranice tiče ovo je glavni element na koji se obraća pozornost. Unutar ovog elementa se kreira sav sadržaj HTML dokumenta, odnosno, stranice koju on reprezentira.

3.3.2. Struktura HTML elementa

Svaka HTML oznaka koja u paru kreira HTML element počinje znakom < (manje od), nakon toga slijedi ime oznake, a završava znakom > (više od). Zatvarajuća HTML oznaka kreira se na isti način kao i otvarajuća, ali se nakon početnog znaka < dodaje i kosa crta / (*eng. slash*), te onda slijedi ime oznake popraćeno znakom >. Na taj način strukturirana standardna oznaka označava jedan HTML element.[9] Primjerice:

```
<div></div>
```

Osim standardnih HTML elemenata postoje i tako zvani “samozatvarajući“ elementi. Takvi element se kreiraju na slijedeći način. Element počinje jednako kao i standardni element oznakom < (manje od), nakon čega slijedi ime oznake, ali završava / (kosa crta) popraćenom znakom > (veće od).[9] Primjerice:

```
<input value='Pozdrav svijetu'/>
```

Svaki HTML dokument moguće je kreirati u bilo kojem uređivaču teksta (*eng. text editor*). No unatoč tome za brže i jednostavnije korištenje preporučljivo je koristiti neki od naprednijih integriranih razvojnih okoliša (*eng. Integrated Development Enviroment*) ili skraćeno IDE. Primarni razlog tomu je da bi se dobio pristup različitim mogućnostima od kojih ću izdvojiti programsku potporu za nadopunjavanje teksta (*eng. IntelliSense*) pomoću koje IDE sugerira i pomaže u pisanju samog koda, te naravno automatska recenzija koda (*eng. CodeReview*) pomoću koje IDE provjerava da li je sve dobro napisano, te da li je na primjer svaki standardni HTML element zatvoren na način naveden ranije u odlomku.

3.3.3. Primjer HTML dokumenta

Za kraj još preostaje primjer jednog HTML dokumenta formatiranog prema smjernicama navedenim u prethodnom odlomku:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naziv stranice</title>
  </head>
  <body>
    <div>
      <p>Ovdje se unosi sadržaj stranice.</p>
      <input type='text' value='Ovo je samozatvarajuci element' />
    </div>
  </body>
</html>
```

Kao što je vidljivo u ovom primjeru HTML dokumenta su prikazana sva načela koja su ranije spomenuta. Na početku je označeno da se radi o dokumentu tipa HTML. Nakon toga je deklariran početak samog HTML dokumenta oznakom:

```
<html>
```

Kao što je ranije spomenuto, otvoreno je zaglavlje dokumenta u kojem je deklariran naslov HTML dokumenta koji je u ovom slučaju glasi: “Naslov stranice“:

```
<title>Naziv stranice</title>
```

te je nakon deklaracije naslova zatvoreno zaglavlje HTML dokumenta. Iduće slijedi oznaka za početak tijela (*eng. body*) HTML dokumenta:

```
<body>
```

U njemu se nalazi sav sadržaj stranice. Sadržaj stranice se u ovom primjeru sastoji od dva elementa koji se nalaze unutar jednog blok elementa oznake:

```
<div>
```

Prvi element je paragraf element čiji je sadržaj glasi: “Ovdje se unosi sadrzaj stranice“:

```
<p>Ovdje se unosi sadržaj stranice.</p>
```

Drugi element je još jedan primjer samozatvarajućeg elementa sa atributom koji ukazuje da će sadržaj unutar ovog elementa biti tekstualnog tipa:

```
<input type='text' value='Ovo je samozatvarajuci element' />
```

Nakon spomenuta dva elementa zatvara se blok element pripadnom oznakom za zatvaranje standardnog HTML elementa koja je u ovom slučaju slijedeća:

```
</div>
```

Također se zatvara element tijela HTML dokumenta oznakom:

```
</body>
```

Ukoliko bi postojala javascript datoteka za koju bi bilo potrebno dodati referencu, ovo je mjesto gdje bi se takvo što dodalo, primjerice:

```
<script src="ime_javascript_datoteke.js"></script>
```

Nakon elementa tijela, ali prije zatvaranja HTML dokumenta. Za kraj zatvaramo i HTML dokument oznakom:

```
</html>
```

3.4. Komentari

Komentari u HTML jeziku nemaju direktnu ulogu prilikom prikaza HTML datoteke, štoviše to je njihova glavna značajka - činjenica da se ne prikazuju. U standardnom prikazu stranice komentari nisu vidljivi, vidljivi su samo kada se otvori izvor (*eng. source*) neke stranice ili kada se HTML datoteku otvori u nekom uređivaču teksta. Oni služe za organizaciju koda te lakše snalaženje u njemu i kao takvi imaju značajnu ulogu. Primjerice pomoću komentara je moguće dokumentirati zadaću koju ima pojedini dio HTML koda. Također su korisnik prilikom samog razvoja nekog HTML dokumenta jer pomoću njih je moguće trajno ili privremeno isključiti dio HTML koda u svrhu lakšeg pronalaska greške u kodu.

3.4.1. Struktura komentara

Komentar ima set pravila sličan HTML elementima. Svaki komentar počinje slijedom znakova < (manje od), zatim znak ! (uskličnik) i nakon njih slijede dva znaka – (minus):

```
<!--
```

Tim slijedom znakova komentar je otvoren, a zatim slijedi proizvoljan tekst. Bitno je napomenuti da kada je komentar jednom otvoren jedini način da se zatvori je njegova zatvarajuća oznaka, što znači da komentar može zauzimati proizvoljan broj redova bez potrebe za otvaranjem i zatvaranjem komentara u svakom redu. Nakon proizvoljnog teksta zapisuje se slijed znakova koji označavaju komentar zatvorenim. Sada kreće od dvije oznake – (minus), te nakon njih slijedi > (veće od):

```
-->
```

Nakon tog slijeda oznaka komentar je zatvoren i sve poslije toga će biti prikazano na stranici.

Primjer komentara:

```
<!—Ovo je komentar u kodu -->
```

3.4.2. Meta elementi

Meta elementi nisu komentari kao takvi, ali s obzirom da se također ne prikazuju prilikom pregleda stranice našli su svoje mjesto u ovom poglavlju.

Meta elementi su dijelovi HTML stranice koje upotrebljavaju mrežne tražilice (*eng. search engine*) da bi zapisale informacije o nekoj stranici. Ovi elementi sadrže ključne riječi, opis, informaciju o vlasništvu, naziv stranice itd. Oni su među mnogim stvarima koje ispituju mrežne tražilice kada "gledaju" stranicu. Iako nije obavezno, vrlo ih je korisno upotrebljavati, pogotovo ako se želi postići dobra pozicija na mrežnim tražilicama. Ako se izradi mrežna stranica i registrira se jedinstveni lokator resursa (*eng. Uniform Resource Locator*) ili skraćeno URL kod mrežnih tražilica, one će posjetiti stranicu i pokušati je indeksirati. Svaka mrežna tražilica funkcionira malo drugačije, i svaka drugačije ocjenjuje pojedine elemente mrežne stranice. Npr. Altavista daje prednost opisnoj oznaci (*eng. description*), a Inktomi indeksira oboje, i tekst stranice, kao i meta elemente. Drugi mrežni pretraživači poput Exactseek su pravi pretraživači meta elemenata, tako da ako stranica ne sadrži naziv (*eng. title*), i opisni meta element. Naravno, ne rade svi mrežni pretraživači na ovaj način. Neki daju prednost sadržaju. Mrežni pretraživači u obzir uzimaju više od 100 stvari kada razmatraju neku stranicu. Najveći razlog zašto mnoge tražilice ne daju toliku važnost meta elementima je zbog malih beznačajnih dosadnih poruka (*eng. Small pointless anoying message*) ili skraćeno SPAM. Ljudi su se dosjetili da na svoje stranice stave mnoge ključne riječi koje nemaju veze sa sadržajem stranice samo da bi dobili što više posjeta, te da bi njihova stranica što više kotirala na mrežnim pretraživačima. Nakon nekog vremena neke tražilice su prestale gledati meta elemente, služili su im uglavnom samo kao potvrda da bi bili sigurni da odgovaraju onome što se nalazi na stranici. Kada su ključne riječi u meta elementima potpuno nevezane za sadržaj stranice, neke mrežne tražilice će kazniti tu stranicu sa guranjem te stranice na dno rezultata pretraživanja.[9]

3.5. Dodavanje zvuka, slike i multimedije

Za kraj će biti pokazana nešto naprednija praktična primjena HTML jezika kroz primjere u kojima će biti demonstrirano kako dodati zvuk, sliku i multimedijski zapis unutar HTML datoteke. Postoje tri načina kojim se multimedijski sadržaj uključuje na neku mrežnu stranicu. Kao dodaci mrežnog preglednika (*eng. plug-in*), ActiveX kontrole i Java umetak (*eng. applet*). Važno je napomenuti da se multimedijski zapis pridodan nekoj HTML stranici samo emitira sa te mrežne stranice, njegovo izvorište ostaje i dalje jednako. Štoviše kada bi se izvorište multimedijskog sadržaja promijenilo ili ako bi se zapis obrisao sa svog mjesta izvorišta, taj multimedijski zapis pridodan HTML stranici ne bi više radio. Drugim riječima, zapravo se ne

dodaje multimedijski zapis na HTML stranicu, nego samo referenca na izvorište multimedijskog zapisa.

3.5.1. Dodaci mrežnog preglednika

Dodatak mrežnog preglednika je program inkorporiran unutar mrežnog preglednika pomoću kojeg se proširuju mogućnosti samog mrežnog preglednika vezane za obradu multimedijskih zapisa, neovisno o njihovom tipu. Bili oni zvučni zapisi, video zapisi, grafički zapisi itd. Postoji razlika između dodatka mrežnog preglednika i pomoćnih aplikacija (*eng. helper application*) koje se otvaraju unutar posebnog prozora, te rade neovisno o mrežnom pregledniku. Za korištenje dodataka mrežnog preglednika unutar HTML stranice, na raspolaganju su trenutno tri vrste HTML oznaka. Kronološki gledajući postoji stara oznaka:

```
<embed>
```

Ako se gleda prema standardu koji je uveo HTML 4.01, tada imamo oznaku:

```
<object>
```

Te postoje trenutne, najnovije oznake koja su uvedene kao dio HTML5 standarda:

```
<video>
<audio>
```

3.5.2. ActiveX kontrole

ActiveX je jedna od glavnih tehnologija koje se koriste u softverskom inženjerstvu temeljenom na komponentama. ActiveX je podržan u mnogim brzim tehnologijama za razvoj aplikacija s ciljem da omogući programerima aplikacija da ugrade ActiveX kontrole u svoje proizvode.[9]

Mnoge aplikacije sustava Microsoft Windows, uključujući mnoge od onih iz samog Microsofta, kao što su Internet Explorer, Microsoft Office, Microsoft Visual Studio i Windows Media Player, koriste ActiveX kontrole kako bi izgradili svoj skup značajki i dodali vlastitu funkcionalnost, te time postali dio ActiveX kontrola sa krajnjim ciljem – olakšanom ugradnjom u druge aplikacije. Internet Explorer također omogućava ugrađivanje ActiveX kontrola na mrežnim stranicama.

3.5.3. Java umetak

Java umetak je mala aplikacija koja je napisana u Java programskom jeziku ili nekom drugom programskom jeziku koji se kompilira (*eng. compile*) u Java bajt kod (*eng. bytecode*) i dostavlja korisnicima u obliku Java bajt koda. Korisnik pokreće Java umetak sa mrežne stranice, a umetak se izvršava unutar Java virtualnog stroja (*eng. Java Virtual Machine*) ili

skraćeno JVM u postupku koji je odvojen od samog mrežnog preglednika. Java umetak se može pojaviti u okviru mrežne stranice, novog prozora aplikacije ili samostalnog alata za testiranje umetka.[9]

3.5.4.Zvuk i multimedija

U nastavku je pokazano kroz primjere kako zapravo dodati zvuk i multimediju. U ovim primjerima koji slijede se koriste sva tri spomenuta načina zapisivanja ovisno o standardima.

Primjer starog HTML standarda (EMBED oznaka):

```
<html>
  <head>
    <title>Primjer EMBED oznake</title>
  </head>
  <body>
    <p>Umetanje zvuka pomoću EMBED elementa</p>
    <embed src='ime_zvucnog_zapisa.mp3 height="60" width="120">
  </body>
</html>
```

Kao što je ranije spomenuto ovo je najstariji način dodavanja multimedijskog sadržaja. U ovom primjeru su uz dodavanje same datoteke postavljeni i neki atributi, oni ukazuju na to koliko će mjesta u pikselima (*eng. pixels)* zauzimati prozor unutar kojeg će se nalaziti multimedijski zapis.

Primjer HTML 4.01. standarda (OBJECT oznaka):

```
<html>
  <head>
    <title>Primjer OBJECT oznake</title>
  </head>
  <body>
    <p>Umetanje videa pomoću OBJECT elementa</p>
    <object data='ime_vizualnog_zapisa.mp3 height="60" width="120">
  </body>
</html>
```

Ovaj način je vrlo sličan starijem načinu, jedina novina koje je uvedena, osim promjene imena HTML oznake je ime atributa, nije više “src“ nego je “data“.

Primjer HTML5 standarda (VIDEO i AUDIO):

```
<html>
  <head>
    <title>Primjer VIDEO i AUDIO oznaka</title>
  </head>
  <body>
    <p>Umetanje videa pomoću VIDEO elementa</p>
      <video src="ime_video_zapisa.mp3" width="640" height="480"
      controls></video>
    <p>Umetanje zvuka pomoću AUDIO elementa</p>
    <audio src="ime_zvucnog_zapisa.mp3" autoplay='true'>
  </body>
```

```
</html>
```

U ovom primjeru je vidljivo da se za zvuk i video koriste različiti HTML elementi, te shodno tome imaju i dodatne atribute koje je moguće koristiti. primjerice prilikom dodavanja video zapisa, dodan je atribut “controls“:

```
<video src='ime_video_zapisa.mp3' width='640' height='480'controls></video>
```

pomoću spomenutog atributa “controls“ omogućeno je korisniku stranice upravljanje priloženim videom. Također je vidljivo na primjeru dodavanja zvuka da postoji atribut “autoplay“, te da je njegova vrijednost postavljena na istinito (*eng. true*):

```
<audio src="ime_zvucnog_zapisa.mp3" autoplay='true'>
```

pomoću atributa “autoplay“ je omogućena opcija da se zvučni zapisa sam pokrene kada korisnik dođe na stranicu.

3.5.5.Slike

Ukoliko bi mrežne stranice bile sačinjene samo od tekstualnog sadržaja i dalje bi bile tehnološki impresivne, ali bez prisustva slika zasigurno ne bi bile toliko zabavne i bliske velikom spektru ljudi kao što su danas. Dva grafička formata koji se najviše koriste u praksi su format grafičke razmjene (*eng. Graphic Interchange Format)* skraćeno GIF i zajednička skupina fotografskih stručnjaka (*eng. Joint Photographic Experts Group*) ili skraćeno JPEG. JPEG format se primarno koristi za prikaz realističnih slika fotografske kvalitete. Dok se GIF primarno koristi za slike sa nešto manjim spektrom boja, primjerice: zastave (*eng. banner)*, animacije, navigacijske ikone itd,. Među mrežnim dizajnerima veliku je popularnost stekao i format prijenosna mrežna grafika (*eng. Portable Network Graphics)* ili skraćeno PNG koji je kreiran kako bi unaprijedio i u konačnici zamijenio GIF format. [9]

Sve slike, bez obzira na njihov format ubacuju se u HTML datoteku koristeći istu oznaku:

```
<img>
```

S obzirom da element za prikaz slike zahtjeva atribut “src“, tada se :

```
<img src="naziv_slike.jpeg">
```

promatra kao cjeloviti element, tj. on nema svoju zatvarajuću oznaku. Element za ubacivanje slike je, kako je ranije spomenuto, samozatvarajući HTML element. Za kraj slijedi primjer ubacivanja slike:

```
<html>
  <head>
```

```
    <title>Primjer ubacivanja slike</title
  </head>
  <body>
    <img src='naziv_slike.jpg'/>
  </body>
</html>
```

4. Lokalno spremište

Lokalno spremište (*eng. Local storage*) je jedno od tri lokalna spremišta podataka koji su glavna tema ovog rada. U radu se krenulo baš sa lokalnim spremištem kao svojevrsnim uvodom u spremišta podataka. Primarno jer je, što se tiče uporabe u praksi, lako povući paralelu sa svima poznatim kolačićima (*eng. Cookies).* Imajući to na umu za početak se kreće, kao i uvijek, sa nečim općenitim. Tako će se i ovdje krenuti sa definiranjem što je to uopće lokalno spremište, kako radi i zašto nam je potrebno. Nakon toga povući će se ranije spomenuta paralela sa kolačićima, te istaknuti sličnosti, ali i razlike. Zatim se fokus prebacuje na sigurnost koja je u današnje vrijeme vrlo bitan aspekt svake mrežne tehnologije. Nakon sigurnosti vrijeme je za upoznavanje sa događajima lokalnog spremišta. Njihovo poznavanje je bitno za razumijevanje životnog ciklusa (*eng. lifecycle)* lokalnog spremišta. Kada se utvrde osnove vrijeme je za konkretne primjere iz prakse s kojima ću pokušati pokazati zašto danas velika većina mrežnih (*eng. web)* programera bira upravo lokalno spremište podataka kao glavnog saveznika prilikom razvoja (*eng. development)* mrežnih aplikacija.

4.1. Što je lokalno spremište

HTML5 lokalno spremište je komponenta aplikacijskog programerskog sučelja (*eng. interface)* mrežne pohrane (*eng. web storage)*. To je metoda kojom mrežne stranice (*eng. website)* lokalno pohranjuju uređene parove – ključ/vrijednost unutar klijentskog mrežnog preglednika. Vrlo slično kolačićima, spremljeni podaci postoje čak i kada se kartica (*eng. tab)* preglednika zatvori, nastavi sa korištenjem mrežnog preglednika na neku drugu stranicu, također isto vrijedi i za napuštanje trenutne kartice preglednika pa čak i za zatvaranje mrežnog preglednika. Za razliku od kolačića ti podaci se ne prenose na udaljenog mrežnog poslužitelja (*eng. remote web server),* osim ako se ne pošalju tamo ručno. Bitno je napomenuti da, s obzirom da je HTML5 lokalno spremište nativno integrirano (*eng. natively integrated)* u mrežne preglednike ono je dostupno svima bez potrebe za dodacima preglednika treće strane (*eng. third party plug-in)*. [10]

Lokalno spremište uglavnom se koristi za pohranu i dohvaćanje podataka u HTML stranica sa iste domene (*eng. domain)*. Čak i nakon ponovnog pokretanja preglednika podaci se mogu vratiti iz svih prozora u istoj domeni. Jedni načini da se podaci iz lokalnog spremišta izgube su ručno brisanje, uklanjanje mrežnog preglednika ili uklanjanje cijelog operacijskog sustava (*eng. Operating System)*. Ova vrsta spremišta mrežnim programerima nudi brojne mogućnosti prilikom izrade mrežnih stranica i aplikacija.

Verzije najpopularnijih mrežnih preglednika koje podržavaju mogućnost korištenja lokalnog spremišta su sljedeće: Firefox verzija 3.5 i noviji, Internet Explorer verzija 8.0 i noviji, Chrome verzija 4.0 i noviji, Safari verzija 4.0 i noviji, Mobile Safari, Opera verzija 10.5 i noviji i nativni mrežni preglednik na Android verziji 2.0 i novijim.

4.1.1. Struktura i pristupačnost

HTML5 lokalno spremište strukturirano je uređenim parovima ključ/vrijednost (*eng. key/value),* za razliku od drugih baza koje se primjerice koriste od strane transakcijski orijentiranih aplikacija (*eng. transaction oriented applications)*. Podaci se pohranjuju na temelju imenovanog ključa (*eng. named key), a* jednako tako se i dohvaćaju – koristeći taj ključ. Za razliku od klasične baze podataka podacima se pristupa direktno, te nema potrebe za SQL (*eng. Structured Query Language)* upitima*.* Spomenuti ključ je u obliku niza znamenaka. Podaci koje spremamo mogu biti bilo koje vrste dok god su podržani od strane JavaScript jezika. Podaci se generalno spremaju kao niz znamenaka. Ako korisnici žele spremati i dohvaćati druge tipove podataka moraju koristiti funkcije[10], primjerice:

```
parseFloat()
```

koji se koristi za decimalne brojeve ograničenog broja decimalnih mjesta ili primjerice:

```
parseInt()
```

koji se koristi za cijele brojeve. Uporaba funkcija je nužna kako bi se prebacili kompatibilni podaci u određeni tip podatka koji je potreban za nastavak kodiranja, tj. neke daljnje operacije koje se planiraju izvršiti nad njima.

Iz Javascript koda, moguće je pristupiti HTML5 lokalnom spremištu preko objekta lokalnog spremišta nad globalnim prozorom objekta. Lokalno spremište pohranjuje podatke bez datuma i vremena isteka. Podaci se nikada sami ne brišu, ostaju spremljeni čak i nakon zatvaranja mrežnog preglednika, te im se može pristupiti u bilo kojem trenutku. S obzirom da se sve odvija na strani klijenta (*eng. client side)*, pohranjeni podaci se temelje na mrežnom pregledniku koji se koristi. Važno je napomenuti da ukoliko korisnik koristi više mrežnih preglednika, svaki mrežni preglednik koristi svoje lokalno spremište. Zasebna lokalna spremišta pojedinih mrežnih preglednika ne znaju za lokalna spremišta drugih mrežnih preglednika, te nemaju mogućnost komunikacije i razmjene informacija između sebe.

4.2. Razlika između lokalnog spremišta i kolačića

Kao što je već ranije spomenuto HTML5 lokalno spremište uvelike nalikuje kolačićima zbog svoje primjene. Primarno jer se oba spomenuta mehanizma mogu koristiti za pohranu

podataka u mrežnom pregledniku između HTTP (*eng. HyperText Trasnfer Protocol)* zahtjeva, ali postoji i razlika između HTML5 lokalnog spremišta i kolačića. U nastavku će prvo biti objašnjeno što su to kolačići, te kako ih koristiti, a tek onda će biti govora o razlikama između lokalnog spremišta i kolačića.

4.2.1.Kolačići

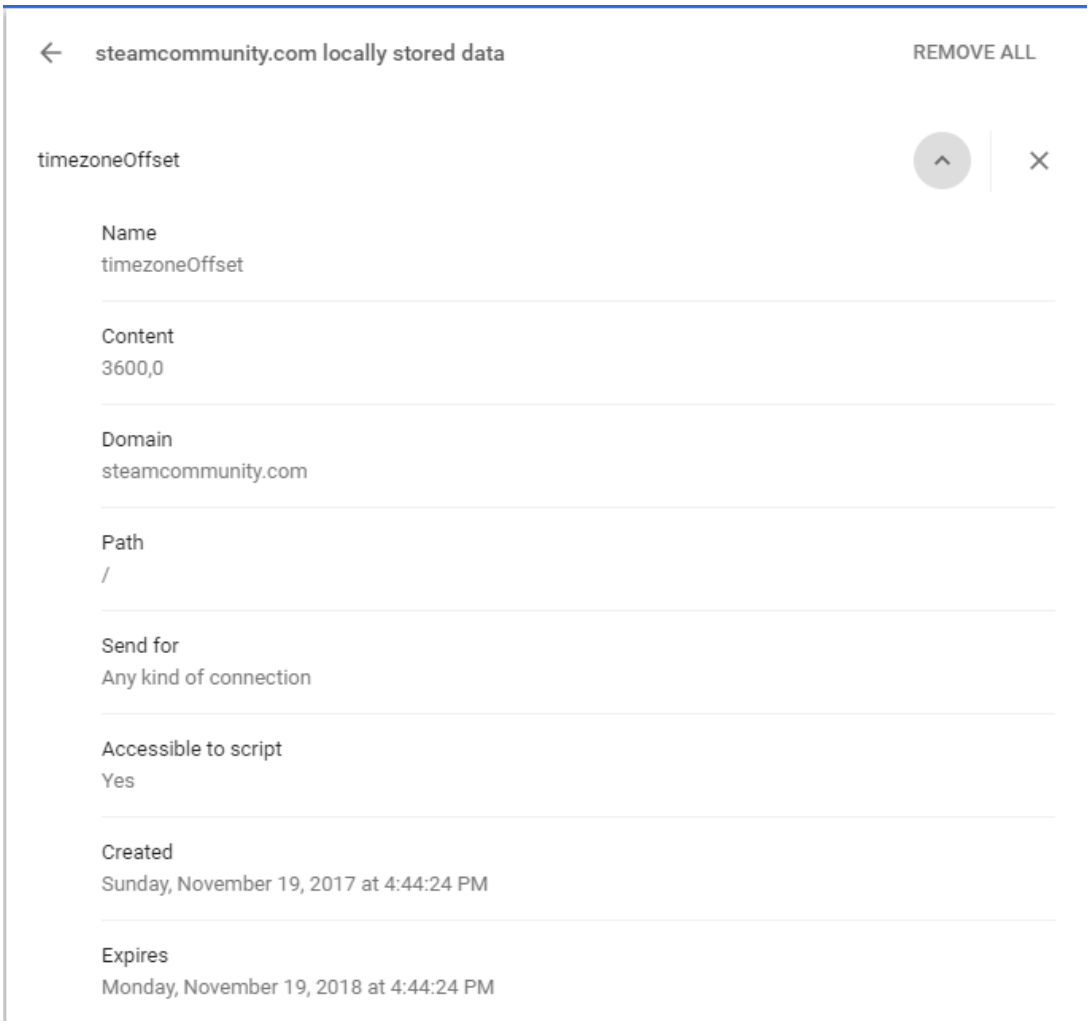
Kolačići (*eng. Cookies)* su obično male tekstualne datoteke koje sadrže dane identifikacijske oznake pohranjene u direktoriju mrežnog preglednika ili podmape (*eng. subfolder)* programskih podataka. Kolačići se izrađuju u trenutku kada korisnik koristi mrežni preglednik da pristupi nekoj mrežnoj lokaciji koja upotrebljava kolačiće za praćenje kretanja korisnika unutar te mrežne lokacije, tj. služi kao pomagač (*eng. helper)* da korisnik nastavi na mjestu gdje je stao, zapamti korisničke podatke potrebne za prijavu, odabir teme, postavke i ostale funkcije prilagodne mrežne lokacije. Mrežna stranica sprema odgovarajuću datoteku, sa istom identifikacijskom oznakom, kao i onu koja je postavljena u korisničkom mrežnom pregledniku, te na taj način omogućuje praćenje informacija vezanih za kretanje unutar mrežne stranice i ima pristup svim informacijama koje korisnik dobrovoljno daje na uvid prihvaćanjem odricanjem odgovornosti za kolačiće (*eng. cookie disclaimer)* i politikom kolačića (*eng. cookie policy)*.[11]

Struktura kolačića

Kolačići obično ne sadrže mnogo informacija, osim jedinstvenog lokatora resursa (*eng. Uniform Resource Locator*) ili skraćeno URL mrežne stranice koja je stvorila kolačić, vremenskog trajanja kolačića i nasumce odabranog broja (*eng. random number)*. Zbog male količine podataka koje kolačić sadrži, najčešće se ne može koristiti za otkrivanje identiteta korisnika ili osobnih identifikacijskih podataka. No unatoč tome marketing postaje sve sofisticiraniji, te nije neobična pojava da se kolačići agresivno koriste za stvaranje profila ovisno o navikava korisnika prilikom pretraživanja mrežnih stranica.[11]

Vrste kolačića

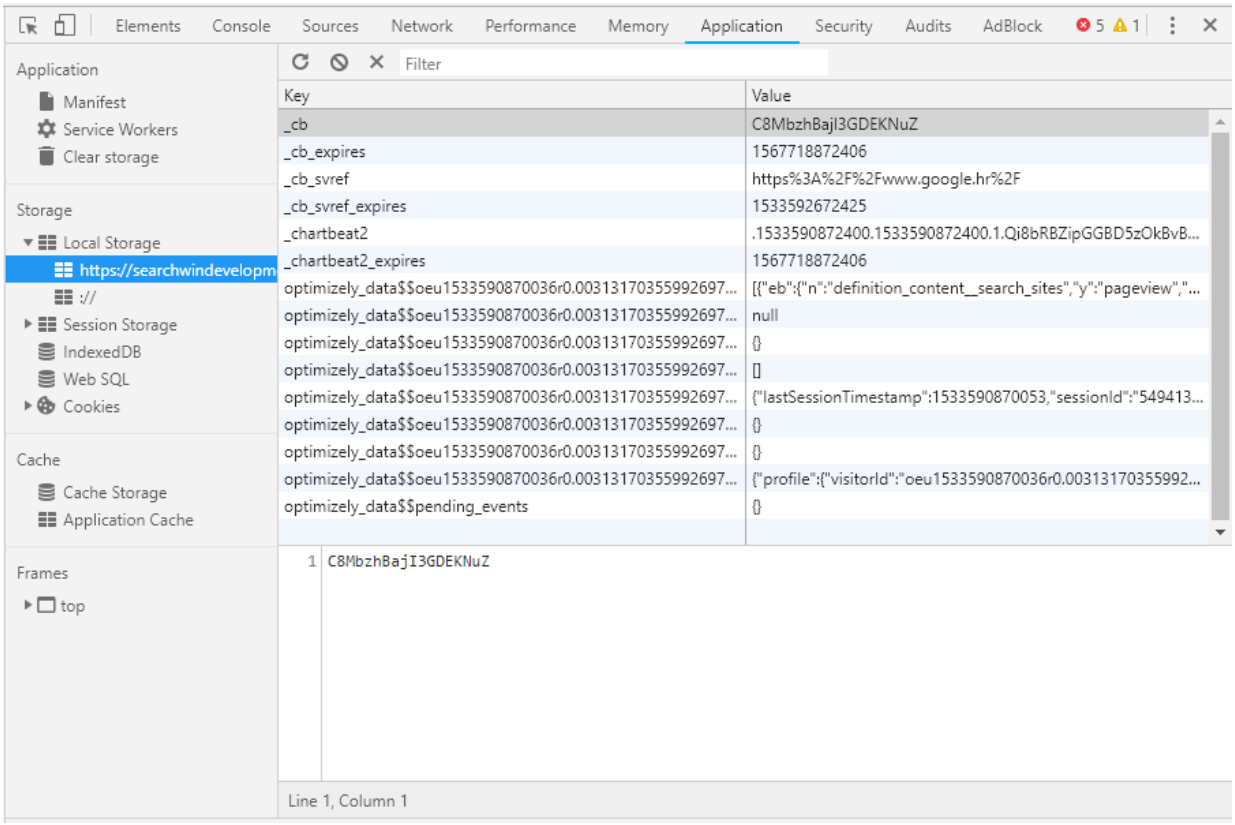
Postoje dvije vrste kolačića: kolačići sesije (*eng. session cookies)* i trajni kolačići (*eng. persistent cookies)*. Kolačići sesije stvaraju se privremeno u podmapi mrežnog preglednika dok god je korisnik u posjeti mrežnoj stranici. Kada korisnik napusti mrežnu stranicu kolačić sesije se briše. S druge strane, trajne datoteke kolačića ostaju u podmapi korisnikovog mrežnog preglednika i ponovno se aktiviraju nakon što korisnik ponovno posjeti mrežnu stranicu koja je stvorila kolačić. Trajni kolačić ipak nema neograničeno trajanje, on se briše iz podmape mrežnog preglednika nakon isteka vremena postavljenog u polju trajanje (*eng. duration)* prilikom kreiranja samog kolačića.[11]



Slika 2: Prikaz jednog kolačića u Chrome mrežnom pregledniku

4.2.2.Usporedba

Kao što je već spomenuto, HTML5 lokalno spremište slično je kolačićima jer se oba mehanizma mogu koristiti za pohranu podataka u pregledniku između HTTP zahtjeva. Ali postoji razlika između HTML5 lokalnog spremišta i kolačića. Kolačići su mali dijelovi podataka koje poslužitelj može pohraniti u pregledniku. Mrežni preglednik šalje kolačić, zajedno sa svim budućim HTTP zahtjevima poslužitelju koji postavlja kolačić. Kolačići ne smiju biti veći od ukupno 4KB. HTML5 lokalno spremište postavljen je putem JavaScript skripte izvršene u mrežnom pregledniku. HTML5 lokalne značajke pohrane nikada se ne šalju na bilo koji poslužitelj - osim ako se izričito ne kopiraju i dodaju u AJAX (*eng. Asynchronus JavaScript and XML)* zahtjev (*eng. request*). HTML5 lokalno spremište može pohraniti negdje između 2 MB i 10 MB podataka u mrežnom pregledniku (po podrijetlu - naziv domene). Točno koliko je podataka dopušteno ovisi o mrežnom pregledniku. Najčešća je granica od 5 MB do 10 MB.



Slika 3: Prikaz lokalnog spremišta u Chrome mrežnom pregledniku

4.3. Sigurnost lokalnog spremišta

Ovo poglavlje će primarno pokušati objasniti prednosti, ali i mane HTML5 lokalnog spremišta sa aspekta sigurnosti. Također će biti prikazan potencijalne vrste napada koji mogu biti opasni i štetni prilikom korištenja lokalnih spremišta podataka unutar mrežnih aplikacija. Za kraj će biti pokazani implementacijski rizici koji su prisutni.

4.3.1. Napadi podvale sustava domene

Zbog potencijalnih napada podvale sustava domene (eng. DNS Spoofing attacks), ne može se jamčiti da je domaćin (eng. host) koji tvrdi da je u određenoj domeni zaista iz te domene. Kako bi se to ublažilo, stranice mogu koristiti TLS (eng. Transport Layer Security). Stranice koje koriste TLS mogu biti sigurne da samo korisnici, aplikacije koje rade u ime korisnika i ostale strance koje koriste TLS sa pripadajućim certifikatima koji ih identificiraju kao iste domene mogu pristupiti njihovim područjima za pohranu.[12]

4.3.2. Križni napadi direktorija

Različiti autori koji dijele ime domaćina (eng. host name), primjerice korisnici koji dijele sadržaj (eng. hosting content) na geocities.com, dijele jedan lokalni objekt za pohranu. Nema mogućnosti ograničavanja pristupa putem naziva staze (eng. pathname). Od autora na zajedničkim računalima se stoga zahtjeva da izbjegavaju korištenje ovih mogućnosti (eng. features), jer bi drugim autorima čitanje i brisanje tih podataka bio u potpunosti trivijalan proces. Važno je napomenuti da čak i ako je dostupna mogućnost ograničena za putanje (eng. pat-restriction), uobičajeni sigurnosni model DOM (eng. Document Object Model) skriptiranja učinio bi zaobilaženje te zaštite trivijalnim procesom.[12]

4.3.3. Implementacijski rizici

Dva primarna rizika prilikom implementacije ovih trajnih mogućnosti pohrane omogućuju neprijateljskim mrežnim lokacijama da pročitaju informacije s drugih domena te ostavljaju neprijateljskim mrežnim mjestima da pišu informacije koje se zatim čitaju s drugih domena.[12]

Dozvoljavanje mrežnih lokacija trećih strana da čitaju podatke koji se ne bi trebali čitati iz njihove domene i time uzrokuju propuštanje informacija, primjerice, korisnikov popis želja za kupnju na jednoj domeni lako bi mogla koristiti druga domena za ciljano oglašavanje ili preko povjerljivih dokumenata koji su pohranjeni na mrežnom mjestu za obradu teksta, korisnik je u mogućnosti pregledati mrežno mjesto konkurentskog poduzeća.[12]

Dozvoljavanje mrežnih mjesta trećih strana da napišu podatke na stalno trajno spremište drugih domena može rezultirati podvalom (eng. spoofing) informacija, što je jednako opasno. Primjerice neprijateljsko mrežno mjesto moglo bi dodati stavke korisnikovoj listi želja za kupnju ili bi tako moglo postaviti identifikator sesije korisnika poznatoj identifikacijskoj oznaci koju bi onda moglo koristiti za praćenja akcija korisnika.[12]

4.4. Događaj lokalnog spremišta

Poglavlje vezano za događaje HTML5 lokalnog spremišta podataka će za početak objasniti što je to uopće događaj u ovom kontekstu, zašto su događaji bitni i za što se koriste. Nastavit će se sa odgovorima na pitanja, što su to slušatelji događaja (*eng. event listener)* i koja je njihova uloga, te za što se koriste. Nakon utvrđivanja osnova fokus će se prebaciti na događaje specifične za lokalno spremište podataka (*eng. local storage event).* Ovdje će biti govora o tome zašto su nam događaji lokalnog spremišta podataka bitni za uporabu lokalnog spremišta podataka. Za kraj malo praktičnog dijela potkrepljenog primjerima iz prakse.

4.4.1. Događaj

Događaj je radnja ili pojava prepoznata od strane sustava. U većini slučajeva ta radnja ili pojava je asinkronog karaktera sa vanjskim izvorištem, ali sustav zna baratati njome. S obzirom da je događaj entitet koji obuhvaća akciju i kontekstualne varijable koje imaju sposobnost pokretanja događaja može se koristiti akrostična mnemografija nekog događaja kao "**E**xecution **V**ariable **E**ncapsulating **N**amed **T**rigger" – EVENT, kako bi se pojasnio koncept događaja. Računalni događaji mogu biti generirani ili pokrenuti od strane sistema, od strane korisnika ili na neki treći način. Najčešće se događaji sinkroniziraju sa programskim tokom, tj. sustav može imati jedno ili više određenih mjesta gdje se događaji obrađuju. Izvor događaja uključuje korisnika koji može, recimo, komunicirati sa sustavom putem pritiska tipke na tipkovnici ili pomicanja računalnog miša. Drugi izvor događaja može biti uređaj, npr. vremenski brojač (*eng. timer)*. Sustav također može pokrenuti događaj koristeći vlastiti skup događaja u petlji događaja (*eng. event loop),* npr. sa svrhom javljanja da je neka aktivnost završena. Sustav koji mijenja svoje ponašanje kao odgovor na događaje nazivamo sustavom upravljanim događajima (*eng. event-driven system)*. Najčešće se takvi sustavi koriste sa svrhom maksimalne interaktivnosti sa korisnikom[13].

4.4.2. Slušatelji događaja

Slušatelji događaja (*eng. event listener)* su postupci ili funkcije u računalnom programu koji čekaju događaje. Ti događaji mogu biti bili koja akcija od strane korisnika, poput pritiska tipke ili pomicanja računalnog miša. Također mogu biti od strane sustava. Slušatelj je zapravo petlja koja je programirana da reagira na neki ulaz ili signal.[13]

Izraz slušatelj je tipično specifičan za programski jezik Java i skriptni jezik Javascript. U ostalim programskim jezicima takvi postupci ili funkcije se nazivaju upraviteljima događaja (*eng. event handler)*.

Slijedeći primjer iz Javascript skriptnog jezika će pokazati kako vezati proizvoljnu funkciju za neki događaji.

```
document.addEventListener('click', myFunction, false);

function myFunction() {
        console.log('ispis u konzolu');
}
```

U ovom jednostavnom primjeru je pokazano slijedeće. Od trenutka kad je mrežni preglednik učitao HTML datoteku, svaki klik računalnog miša pokrenuti će funkciju nazvanu `myFunction` koja će zapisati u konzolu mrežnog preglednika "ispis u konzolu".

4.4.3. Lokalni događaji

Kada se dogodi izmjena u lokalnom spremištu podataka, mrežni preglednik aktivira događaje pohrane (*eng. storage events)*. Događaj za pohranu se aktivira kada se umetne, ažurira ili izbriše neko svojstvo lokalnog spremišta podataka. Događaj za pohranu se emitira samo u drugim prozorima mrežnog preglednika od strane prozora koji je izvršio izmjenu. Za objekt lokalnog spremišta podataka koji se dijeli kroz različite prozore mrežnog preglednika, ti događaji su vidljivi svim ostalim otvorenim prozorima s istim podrijetlom, tj jednakom kombinacijom protokola i naziva domene. U to ulaze i skočni prozori (*eng. popup window)*, te iframe prozori uključeni u HTML.[12]

U svrhu demonstracije događaja lokalnog spremišta podataka, primarno valja demonstrirati kako priključiti slušatelja događaja. Stoga priključivanje slušatelja događaja lokalnog spremišta se izvodi na slijedeći način

```
window.addEventListener('storage', onStorageEvent, false);

function onStorageEvent(storageEvent){

    alert("storage event");
}
```

Funkciju “onStorageEvent“ u ovom slučaju nazivamo funkcijom slušatelja događaja (*eng. event listener function).* Kao što je vidljivo na primjeru iz prethodnog poglavlja funkcija “addEventListener“ pozivu pridodaje funkciju slušatelja događaja lokalnog spremišta, te tu akciju nazivamo spremišni događaj (*eng. storage event)*. Objekt spremišnog događaja proslijeđen funkciji slušatelja događaja izgleda ovako:

```
StorageEvent {
    target;
    type;
    bubbles;
    cancelable;
    key;
    oldValue;
    newValue;
    url;
    storageArea;
}
```

Svojstva objekta spremišnog događaja su slijedeća. Svojstvo “target“ predstavlja cilj nekog događaja. Svojstvo “type“ označava tip događaja. Svojstvo “bubbles“ govori da li događaj ima tendenciju postati mjehurićav (*eng. bubbles)*. Svojstvo “cancelable“ govori da li je moguće poništiti događaj. Svojstvo “key“ označava ime svojstva ili promjene. Svojstvo “odlValue“ predstavlja staru vrijednost svojstva, tj. vrijednost prije promjene. Svojstvo “newValue“ predstavlja novu vrijednost svojstva, tj. vrijednost nakon promjene. Svojstvo “url“ označava

jedinstvenog lokatora resursa stranice koja je napravila promjenu. Svojstvo “storageArea“ predstavlja o kojem spremištu se radi, u ovom slučaju to je lokalno spremište.

4.5. Uporaba lokalnog spremišta

Za kraj je još preostala samo uporaba lokalnog spremišta podataka. Do sada je pokazano što je događaj, čemu događaj služi, te kako se koristi. Također je bilo spomena o slušateljima događaja. Povrh toga prošlo poglavlje je zaključeno događajima lokalnog spremišta. Ovo poglavlje će biti orijentirano na praktične primjere uporabe lokalnog spremišta podataka kao saveznika prilikom izrade mrežnih aplikacija.

4.5.1. Funkcije objekta lokalnog spremišta podataka

Prije nego se krene sa primjerima iz prakse bitno je razumjeti koje sve metode stoje na raspolaganju prilikom uporabe lokalnog spremišta podataka. Postoji 5 metoda koje se mogu izvršiti nad objektom lokalnog spremišta podataka, a to su `getItem()`, `setItem()`, `removeItem()`, `key()` i `clear()`. U nastavku će svaka od njih biti prikazana na primjeru.

Metoda getItem()

Sljedeći primjer će biti funkcija koja vraća tri podatkovne stavke iz lokalnog spremišta podataka. Zatim ih koristi za postavljanje prilagođenih stilova na stranici. Važno je napomenuti da metoda `getItem()` prilikom poziva prima jedan parametar koji označava ključ zapisa koji se želi dohvatiti iz lokalnog spremišta podataka.

```
function primjeniStilove() {

  var trenutnaBoja = localStorage.getItem('bgcolor');
  var trenutniFont = localStorage.getItem('font');
  var trenutnaSlika = localStorage.getItem('image');

  htmlElem.style.backgroundColor = '#' + trenutnaBoja;
  pElem.style.fontFamily = trenutniFont;
  imgElem.setAttribute('src', trenutnaSlika);
}
```

Kreirana je funkcija `primjeniStilove()` u kojoj su inicijalizirane tri varijable u koje su spremljene vrijdenosti iz lokalnog spremišta podataka koje su dohvaćene uporabom metode `getItem()`. U nastavku koda postavljene su vrijednosti povučene iz lokalnog spremišta podatka u HTML elemente. Konkretno u ovom slučaju postavljena je boja, font i slika.

Metoda setItem()

Sljedeći primjer će biti funkcija unutar koje će se spremiti neke vrijednosti u lokalno spremište podataka. Nastavit će se sa poznatim primjerom, pa će tako na primjeru biti

pokazano kako se postavljaju vrijednosti korištene u primjeru iz prethodnog poglavlja 4.5.1.1. Metoda getItem(). Važno je napomenuti da metoda `setItem()` prilikom poziva prima dva parametra. Ta dva parametra označavaju ključ i vrijednost koju će zapis imati. Poredak parametara je važan, tako da je prvi parametar uvijek označava ključ, a drugi parametar uvijek označava vrijednost zapisa unutar lokalnog spremišta podataka.

```
function postaviStilove() {

        localStorage.setItem('bgColor', 'red');
        localStorage.setItem('font', 'serif');
        localStorage.setItem('image', 'slika.png');
}
```

Kreirana je funkcija `postaviStilove()` u kojoj su pozvane tri metode `setItem()` nad lokalnim spremištem podataka. Metode spremaju u lokalno spremište podataka uređene parove. Prvi uređeni par ima za ključ 'bgColor', a vrijednost 'red'. Drugi uređeni par ima za ključ 'font', a vrijednost 'serif'. Treći uređeni par ima za ključ 'image', a vrijednost 'slika.png'.

Metoda removeItem()

Slijedeći primjer će biti funkcija unutar koje će se ukloniti neke vrijednosti iz lokalnog spremišta podataka. Nastaviti će se u istom kontekstu pa će se tako ukloniti vrijednosti spremljene u lokalno spremište podataka u prošlom poglavlju 4.5.1.2. Metoda setItem(). Važno je napomenuti da metoda `removeItem()` prilikom poziva prima jedan parametar koji označava ključ zapisa koji se želi obrisati iz lokalnog spremišta podataka.

```
function ukloniStilove() {

        localStorage.removeItem('bgColor');
        localStorage.removeItem('font');
        localStorage.removeItem('image');
}
```

Kreirana je funkcija `ukloniStilove()` unutar koje su uklonjena tri zapisa iz lokalnog spremišta podataka koristeći funkciju `removeItem()`. Kao što je vidljivo iz primjera zapisi se uklanjaju koristeći ključ kao identifikator zapisa.

Metoda clear()

Slijedeći primjer će biti namjenom vrlo sličan prethodnom. Kreirati će se metoda koja će jednom linijom obrisati sve zapise u lokalnom spremištu podataka.

```
function ukloniLokalnoSpremiste() {

        localStorage.clear();
}
```

Kreirana je funkcija `ukloniLokalnoSpremiste()` unutar koje je pozvana samo jedna metoda `clear()`. Pomoću nje su uklonjeni svi zapisi unutar lokalnog spremišta podataka. Naravno to

se ne odnosi na cijelo lokalno spremište podataka nego samo na zapise koji su kreirani od strane aplikacije koja je pozvala metodu `clear()`.

Metoda key()

Slijedeći primjer će biti funkcija unutar koje će se dohvatit ključ nekog zapisa iz lokalnog spremišta podataka koristeći indeks kao identifikator. Važno je napomenuti da metoda `key()` prima jedan parametar, a to je u ovom slučaju indeks prema kojem će pokušati locirati zapis unutar lokalnog spremišta podataka.

```
function dohvatiZapisPrekoIndeksa() {

        localStorage.setItem('bgColor', 'red');
        localStorage.setItem('font', 'serif');
        localStorage.setItem('image', 'slika.png');

        var prviZapis = localStorage.key(0);
}
```

Kreirana je funkcija `dohvatiZapisPrekoIndeksa()` unutar koje je su prvo spremljena tri zapisa u lokalno spremište podatka. Nakon toga je dohvaćen prvi zapis preko njegovog indeksa. Prvi zapis je u ovom slučaju zapis koji ima ključ 'bgColor'. Prvi zapis je spremljen u varijablu nazvanu “prviZapis“.

4.5.2. Primjer iz prakse – sistemski gumb “povratak“

Slijedeći primjer uporabe lokalnog spremišta će pokazati kako koristit sistemski gumb za povratak mrežnog preglednika unutar jednostranične aplikacije ili skraćeno SPA (*eng. single page application)*. S obzirom da se radi o jednostraničnoj aplikaciji, gumb za povratak će zapravo vratiti korisnika na lokaciju na kojoj se nalazio prije samog pristupanja stranici, neovisno o tome u koju dubinu stranice je otišao. No uz pomoć lokalnog spremišta podataka postoji mogućnost da se “prevari“ mrežni preglednik, te uz malo koda omogući da sistemski gumb za povratak radi i u slučaj jednostranične aplikacije.

```
$(document).ready(function() {

    window.onpopstate = function(event){

        console.log("kliknuo back");

        switch (localStorage.getItem('currentPage')) {
            case 'home':
                console.log("case home: možeš van");
                break;
            case 'profile':
               profile_back();
                break;
            case 'about':
                about_back();
                break;
```

```
    };

function profile_back() {

        history.pushState('Home', "", "/Home");
        localStorage.setItem('currentPage',"home");
        $('#home_screen').show();
        $('#profile').hide();
    }
function options_back() {

        history.pushState('Home', "", "/Home");
        localStorage.setItem('currentPage',"home");
        $('#options').hide();
        $('#home_screen').show();
    }
$('#profile_tab_navigate').click(function () {

        history.pushState('Profile', "", "/Profile");
        localStorage.setItem('currentPage','profile');

        $('#home_screen').hide();
        $('#profile').show();
}

$('#options_tab_navigate').click(function () {

        history.pushState('Options', "", "/Options");
        localStorage.setItem('currentPage','options');

        $('#home_screen').hide();
        $('#options').show();
}

});
```

Kao što je spomenuto ranije, ovih par linija koda je omogućilo uporabu sistemskog gumba za povratak iako se radi o jednostraničnoj aplikaciji. Sada će biti objašnjen Javascript kod, a nakon toga će biti pokazan i pripadni HTML kod kako bi se dobila bolja perspektiva što je zapravo napravljeno u javascript dijelu.

Na samom početku napisana je linija koda koja do sada nije nigdje prikazana. Sada će biti prikazano koja je njezina uloga, naime kada bi se maknuo ostali kod, izgledalo bi ovako:

```
$(document).ready(function() {});
```

Ovo je zapravo još jedan sigurnosni mehanizam koji je pametno koristiti. Ovdje je zapravo rečeno da funkcija unutar funkcije `ready` neće biti izvršena, niti će se probati izvršiti sve dok se HTML dokument ne učita. Zašto je to bitno? Pa zato jer kod koji se izvršava unutar te funkcije je direktno vezan za HTML dokument i primarno je nužno da HTML datoteka bude učitana.

Nadalje određena je funkcija koja će biti pokrenuta u trenutku kada korisnik pritisne sistemski gumb za povratak. Ta funkcija u sebi sadrži prekidač (*eng. switch)* sa navedenim slučajevima (*eng. case)*. Svi slučajevi osim slučaja 'home' pozivaju prikladne funkcije koje zapravo zaobilaze akciju koja bi bila odrađena pritiskom na sistemsku tipku povratak. Slučaj 'home' to ne radi jer je stranica “Home“ zapravo indeksna stranica i kada se na njoj pritisne povratak ideja je da se napusti stranica.

Za kraj je još preostao prikaz HTML datoteke nad kojom ovaj javascript vrši svoje funkcije. Kao što je ranije spomenuto, ovdje će se vidjeti koja je ideja jednostranične aplikacije i kako se cijela priča povezuje preko javascript skriptnog jezika.

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>test localStorage</title>
</head>
<body>
        <div id=”home_screen”>
                <button id=”profile_navigate”>profile</button>
                <br/>
                <button id=”options_navigate”>options</button>
        </div>

        <div id=”profile” style=”display: none;”>
                <h1>Ovo je profilna stranica</h1>
        </div>

        <div id=”options” style=”display: none;”>
                <h1>Ovo je stranica postavki</h1>
        </div>

</body>
<script src="script.js" type="text/javascript"></script>
</html>
```

Ovo je prikaz male HTML datoteke koja u kombinaciji sa prethodno viđenim javascript kodom omogućuje navigaciju pomoću sistemske tipke nazad mrežnog preglednika. Pa vrijeme je za kratku analizu HTML datoteke. Na samom početku je deklarirano da se radi o HTML datoteci. U zaglavlju HTML datoteke je postavljeno ime, a to je u ovom slučaju “test localStorage“. Nakon toga se otvara tijelo HTML datoteke u kojem postoje tri elementa, svaki od njih ima svoj atribut “id“ na osnovu kojega se vrši identifikacija elemenata u javascript skriptnom jeziku. Prvi element unutar sebe ima i dva <button> elementa, oni predstavljaju gumbe na ekranu. Oni također imaju svoje atribute “id“ iz istog razloga, identifikacije. Kao što je vidljivo, ova HTML datoteka je u potpunosti statična, sve akcije i funkcije se odrađuju unutar javascript datoteke. Također valja primijetiti da druga dva

elementa imaju na sebi atribut “style“ unutar kojega se nalazi mali komadić CSS-a koji zapravo kaže, nemoj ovo prikazati.

Za sam kraj valja napomenuti da je ovdje korišten još jedan sigurnosni mehanizam. Nakon zatvaranja elementa tijela HTML datoteke, ali prije zatvaranja same HTML datoteke, dodana je veza na javascript datoteku. Zašto je ovo sigurnosni mehanizam? Zato što se ostvaruje veza prema javascript datoteci tek nakon što se učitao sadržaj HTML datoteke i tako se osigurava da se javascript kod ne bi izvršio na HTML datoteci prije nego se ta datoteka učita.

5. Sesijsko spremište

Sesijsko spremište (*eng. Session storage*) podataka je jedno od tri lokalna spremišta podataka koji su glavna tema ovog rada. Nakon detaljne analize i praktičnih primjera lokalnog spremišta podataka u HTML5, nastavlja se sa sesijskim spremištem. Sesijsko spremište podataka dijeli većinu svojih osobina sa lokalnim spremištem tako da će većina stvari biti poznata. No važno je istaknuti baš one dijelove u kojima se sesijsko spremište podataka razlikuje od lokalnog spremišta i od kolačića s kojim je povučena paralela sa lokalnim spremištem podataka. Imajući to na umu za polazišnu točku je izabrano nešto općenito o samom sesijskom spremištu podataka. Dakle definicijom što to sesijsko spremište podataka je, kako radi, za što nam je potrebno, te što se do sada koristilo u tu svrhu, tj. koju je to funkcionalnost sesijsko spremište podataka zamijenilo. Zatim se fokus prebacuje na sigurnost koja je uz jako male iznimke zapravo identična sigurnosti lokalnog spremišta podataka, no i dalje bitan aspekt koji valja promotriti i istaknuti. Nakon sigurnosti biti će govora o događajima sesijskog spremišta. U prethodnom poglavlju je pokazano nešto općenito o samim događajima i slušateljima, tako da će se sada taj dio preskočiti, ali i dalje će biti pokazani događaji specifični za sesijsko spremište, kao i njihova praktična primjena. Za kraj je ostavljena sama praktična uporaba sesijskog spremišta. Ovo poglavlje će obilovati stvarnim primjerima iz prakse, te naravno detaljnim objašnjenjima po koracima rada.

5.1. Što je sesijsko spremište

Objekt sesijskog spremišta podataka postoji kao svojstvo objekta prozora u podržavanju različitih mrežnih preglednika, trenutno se radi o preglednicima Firefox verzije 3 i više, Safari verzije 4 i više i Internet Explorer verzije 8 i više. Moguće je staviti podatke na objekt sesijskog spremišta podataka, te ti podaci ostaju vezani za objekt sesijskog spremišta podataka dok god je taj prozor otvoren. Jednako vrijedi i za karticu (*eng. tab)* u mrežnom pregledniku. Važno je napomenuti da čak i ako korisnik krene dalje od stranice koja je pohranjivala podatke i zatim se vratio na stranicu, podaci spremljeni u sesijsko spremište podataka su i dalje dostupni. Svi podaci pohranjeni u sesijsko spremište podataka vezani su za protokol, naziv domaćina (*eng. host)* i priključak same stranice koji je spremio podatke. Također treba istaknuti da samo one stranice koje dijele isti protokol, naziv domaćina i broj priključka (*eng. port*) mogu kasnije pristupiti tim podacima.[10]

Sesijsko spremište podataka je jedinstveno za svaki određeni prozor ili karticu mrežnog preglednika. Ako se pogleda u specifikaciju sesijskog spremišta, tamo je definirano da sesijsko spremište podataka odnosi na kontekst pregledavanja najviše razine. Primjerice,

pretpostavimo da korisnik otvori u mrežnom pregledniku servis za elektroničku poštu Yahoo! Mail i to na dvije različite kartice istog preglednika. Aplikacija tada sprema neke podatke u sesijsko spremište podataka. Podaci s prve kartice nisu dostupni drugoj kartici unatoč činjenici da su protokol, naziv domaćina i priključak potpuno jednaki.

Podaci pohranjeni u sesijsko spremištu podataka spremaju se u parove ključa i vrijednosti gdje su ključ i vrijednost nizovi. Jednako kao i u lokalnom spremištu podataka. Vrijednosti koji nisu proslijeđene u obliku niza znakova automatski se pretvaraju u niz znakova prije samog pohranjivanja u sesijsko spremište.[12]

Podaci u sesijskom spremištu podataka se brišu nakon zatvaranja prozora ili kartice mrežnog preglednika ili ako korisnik manualno zatraži da mrežni preglednik to učini. Također, aplikacija koja je spremila podatke u sesijsko spremište podataka ima mogućnost te podatke i obrisati, za razliku od korisnika koji ručno može obrisati podatke bilo koje aplikacije. Ovo ponašanje, u kombinaciji s povezivanjem podataka s određenim prozorom ili karticom, osigurava nehotično otkrivanje ili pohranjivanje podataka na neodređeno vrijeme.

5.2. Sigurnost sesijskog spremišta

Kao što je spomenuto u uvodu u ovo poglavlje, sigurnost sesijskog spremišta u velikoj većini dijeli karakteristike sa lokalnim spremištem podataka. Dakako postoje mala odstupanja koja su zapravo većinom vezana za trajnost podataka spremljenih u sesijsko spremište podataka. Biti će pokazane mane sesijskog spremišta podataka na dva najozbiljnija primjera prijetnji, koji su obrađeni u poglavlju 4.3. Sigurnost lokalnog spremišta, napad podvale sustava domene i križni napadi direktorija. Za kraj će fokus biti prebačen na razliku između sesijskog spremišta podataka i lokalnog spremišta podataka sa aspekta sigurnosti.

5.2.1.Napadi podvale sustava domene

Kao što je vidljivo u poglavlju 4.3.1.Napadi podvale sustava domene, u sesijskom spremištu podataka se također javlja isti problem gdje je zbog potencijalnih napada podvale sustava domene cijelo sesijsko spremište podataka izloženo riziku uz iznimku što prijetnja postoji samo dok je korisnik prisutan na stranici koja je generirala zapis u sesijsko spremište podataka, jer kako je ranije rečeno, čim korisnik napusti prozor ili karticu preglednika, podaci spremljeni u sesijskom spremištu podataka se brišu. No neovisno o tome, kao i u lokalnom spremištu podataka, primjenjuje se isti način prevencije napada podvale sustava domene, a to je korištenje TLS-a. Na taj način stranice mogu biti sigurne da samo stranice sa pripadajućim certifikatima, prema kojima se jednoznačno definira pripadnost određenoj domeni, mogu pristupiti njihovom sesijskom spremištu.

5.2.2. Križni napadi direktorija

Kao što je navedeno u poglavlju 4.3.2.Križni napadi direktorija, u sesijskom spremištu podataka se također javlja ista sigurnosna prijetnja kao i u lokalnom spremištu podataka. Ovdje se isto tako javlja problem kada različiti autori koji dijele ime domaćina dijele jedan lokalni objekt za pohranu. Kao i u slučaju lokalnog spremišta podataka ovdje je također preporuka koja proizlazi iz službene dokumentacije sesijskog spremišta da autori na zajedničkim računalima izbjegavaju korištenje ove mogućnosti. Jer kao što je navedeno ne postoji niti jedna mogućnost ograničavanja pristupa, čak niti ograničavanje putem naziva staze nije dovoljna zaštita u ovom slučaju.

5.2.3. Implementacijski rizici

Ovdje se također valja osvrnuti na prethodno poglavlje 4.3.3.Implementacijski rizici, u kojemu su opisani implementacijski rizici koji se javljaju prilikom implementacije lokalnog spremišta podataka, te analogno tome i sesijskog spremišta podataka. Također se govori o dva primarna rizika koji se javljaju prilikom implementacije. Prvi je naravno dozvoljavanje mrežnih lokacija trećih strana da čitaju podatke koji se ne bi trebali čitati iz njihove domene, te time uzrokuju propuštanje informacija. Drugi rizik je dozvoljavanje mrežnih mjesta trećih strana da zapisuju podatke na sesijsko ili lokalno spremište podataka drugih domena koje može rezultirati podvalom informacija koje će se onda čitati od strane drugih domene i uzrokovati probleme u integritetu podataka.

5.2.4. Razlika između sesijskog spremišta i lokalnog spremišta sa aspekta sigurnosti

Glavni razlog zašto je sesijsko spremište podataka postalo vrlo popularan alat današnjim mrežnim programerima je upravo onaj aspekt sigurnosti u kojemu se razlikuje od lokalnog spremišta podataka. Ograničavanjem pristupa podacima u jednom prozoru ili kartici mrežnog preglednika, povezivanjem tih podataka putem protokola, domene i priključka, a zatim i brisanjem tih podataka kada je prozor ili kartica zatvorena osigurava da se podacima ne može pristupiti većinom štetnih načina. Dakle primarna razlika između sesijskog spremišta podataka i lokalnog spremišta podataka sa aspekta sigurnosti je činjenica da kada korisnik zatvori prozor ili karticu mrežnog preglednika sesijsko spremište podataka se briše dok lokalno ostaje, ako ga se ne obriše manualno. Tim procesom su drastično smanjene sigurnosne prijetnje prilikom korištenja sesijskog spremišta. Može se reći da su svedene na korisničku nepažnju. Primjer te nepažnje bi bio napuštanje nekog javnog računala kojem imaju pristup i

drugi korisnici bez zatvaranja mrežnog preglednika. Na taj način bi drugi korisnik dobio pristup tuđim privatnim podacima bez njegovog znanja i odobrenja.

5.3. Događaj sesijskog spremišta

Poglavlje vezano za događaje sesijskog spremišta podataka će imati velik broj dodirnih točaka sa poglavljem 4.4.Događaji lokalnog spremišta. S obzirom da su same osnove potrebne za razgovor o ovoj temi pokrivene u prethodnim poglavljima 4.4.1.Događaj i 4.4.2.Slušatelji događaja, u narednom poglavlju neće biti priče o tome. Nastavak teksta će biti primarno baziran na događajima karakterističnim za sesijsko spremište podataka (*eng. session storage event)*. Biti će pokazano zašto su nam događaji sesijskog spremišta podataka bitni za uporabu i baratanje sesijskim spremištem podataka. Za kraj će naravno biti pokazani stvarni primjeri iz prakse.

Kada se dogodi izmjena na sesijskom spremištu podataka, mrežni preglednik aktivira događaje pohrane, kao što je bio slučaj i sa lokalnim spremištem podataka u poglavlju 4.4.3.Lokalni događaj. Jednako tako vrijedi da se sam događaj za pohranu aktivira kada se umetne, ažurira ili izbriše neko svojstvo sesijskog spremišta podataka. Analogno tome također vrijedi da se događaj za pohranu emitira samo u drugim prozorima mrežnog preglednika od strane prozora koji je zaslužan za iniciranje procesa izmjene.[12]

Da bi u potpunosti ovaj sadržaj bio razumljiv treba pokazati kako se priključuje slušatelj događaja sesijskog spremišta podataka.

```
window.addEventListener('storage', onSessionStorageEvent, false);

function onSessionStorageeEvent (storageEvent) {

        alert("session storage");
}
```

Iz ovog primjera je jasno vidljivo da se zapravo samo priključivanje slušatelja događaja sesijskog spremišta podataka uopće ne razlikuje od priključivanja slušatelja lokalnog spremišta podataka. Uz pomoć funkcije `addEventListener()` pozivu je pridodana funkcija slušatelja događaja sesijskog spremišta. Objekt spremišnog događaja sesijskog spremišta podataka proslijeđen funkciji slušatelja događaja izgleda ovako:

```
StorageEvent {
    target;
    type;
    bubbles;
    cancelable;
    key;
    oldValue;
    newValue;
    url;
```

```
    storageArea;
}
```

Kao što je vidljivo iz ovog primjera i primjera spremišnog događaja lokalnog spremišta podataka iz poglavlja 4.4.3.Lokalni događaj, sesijsko spremište podataka i lokalno spremište podataka zapravo dijeli isti spremišni događaj. S obzirom da je svako pojedino svojstvo objašnjeno u poglavlju 4.4.3.Lokalni događaj, ovdje se neće ponovno objašnjavati.

5.4. Uporaba sesijskog spremišta

Za završetak ovog poglavlja će biti govora o uporabi sesijskog spremišta podataka. Zanimljiva stvar kod uporabe sesijskog spremišta podataka je paralela koju je moguće povući sa lokalnim spremištem podataka. Naime kako je viđeno kroz ovo cijelo poglavlje, sesijsko spremište podataka i lokalno spremište podataka imaju mnogo dodirnih točaka u samoj tehničkoj prezentaciji oba spremišta. U nastavku će biti vidljivo da su funkcije objekta sesijskog spremišta podataka istovjetne funkcijama objekta lokalnog spremišta podataka. Jedino što se mijenja je zapravo kontekst u kojem se pojedino spremište podataka koristi. Upravo to će se pokušati dočarati primjerima koji slijede.

5.4.1.Funkcije objekta sesijskog spremišta podataka

Kao što je spomenuto u uvodnom dijelu poglavlja, sesijsko spremište podataka i lokalno spremište podataka dijele iste funkcije objekta spremišta podataka. Valja se prisjetiti kojih su to 5 metoda, `getItem(), setItem(), removeItem(), key()` i `clear()`. U nastavku će svaka od njih biti prikazana na primjeru imajući na umu kontekst u kojem se koriste.[12]

Metoda getItem()

Slijedeći primjer će biti funkcija koja dohvaća zapis iz sesijskog spremišta podataka, te provjerava da li je dobivena vrijednost valjana u svrhu autentifikacije korisnika unutar neke mrežne aplikacije.

```
function validirajKorisnickiId() {

        var userId = sessionStorage.getItem(userId);

        if (userId) {
                //korisnički id je validan
        } else {
                //korisnički id nije validan
        }
}
```

Kreirana je funkcija `validirajKorisnickiId()` u kojoj je inicijalizirana varijabla “userId“, te joj je pridružena vrijednost koja je dohvaćena putem metode `getItem()` iz sesijskog spremišta

podataka koristeći ključ “userId“. Nakon toga varijabla “userId“ se validira, te se izvode akcije ovisno o ishodu validacije. Za ovaj konkretan primjer nije bitno koje će te akcije biti, stoga su samo upisani komentari za svaki ishod.

Metoda setItem()

Slijedeći primjer će biti funkcija pomoću koje će se postaviti zapis u sesijsko spremište podataka korišten u primjeru prethodnog poglavlja. Dobivena vrijednost će zapravo biti rezultat validacije korisničkog imena i lozinke dobivene iz forme za prijavu u neku mrežnu aplikaciju. Valja napomenuti da će biti pokazana samo funkcija koja direktno postavlja zapis u sesijsko spremište podataka, pomoćna funkcija za samu validaciju korisničkog imena i lozinke neće biti pokazana jer nije bitna za ovu demonstraciju.

```
function validateLogin() {

	var korisnickoIme = document.getElementById('username').value;
	var lozinka = document.getElementById('password').value;

	if (validateCredentials(korisnickoIme, lozinka)) {

		var userId = getUserId(korisnickoIme, lozinka);
		sessionStorage.setItem('userId', userId);
	} else {
		//validacija nije uspješna
	}
}
```

Kreirana je funkcija `validateLogin()` u kojoj su inicijalizirane dvije varijable, te su im pridružene vrijednosti korisničkog unosa iz elemenata koji odgovaraju zatraženima. Nakon toga se validiraju vrijednosti spremljene u te dvije varijable, primarno iz sigurnosnih razloga u koje nećemo trenutno ulaziti, a tiču se SQL upita. Ako je validacija uspješna inicijalizira se nova varijabla te se njoj pridružuje vrijednost dobivena iz metode `getUserId()`, te se ta vrijednost sprema u sesijsko spremište podataka kao zapis koji za ključ ima 'userId'.

Metoda removeItem()

Slijedeći primjer biti će funkcija pomoću koje će se ukloniti zapis iz sesijskog spremišta podataka. Kada se primjer stavi u kontekst, u biti će biti prikazano što se dogodi sa zapisom koji je korišten u prethodna dva poglavlja, 'userId' kada se korisnik odluči odjaviti sa stranice.

```
function logout() {

	sessionStorage.removeItem('userId');
}
```

Kreirana je funkcija `logout()` u kojoj je pomoću metode `removeItem()` uklonjen zapis iz sesijskog spremišta podataka koji se identificira prema ključu 'userId'. Ova funkcija će zapravo biti pozvana kao odgovor na akciju korisnika, npr. kada korisnik klikne na gumb odjavi se.

Metoda clear()

Slijedeći primjer će biti funkcija unutar koje će biti pozvana metoda `clear()`. Kao što je objašnjeno u poglavlju 4.5.1.4.Metoda clear() pmoću samo jedne linije koda briše cijelo spremište podataka, u ovom slučaju se radi o sesijskom spremištu podataka koje je generirala određena aplikacija.

```
function ukloniSesijskoSpremiste() {

        sessionStorage.clear();
}
```

Kreirana je funkcija `ukloniSesijskoSpremiste()` unutar koje je pozvana jedna metoda, `clear()`. Kao što je bio slučaj i sa uporabom spomenute metode nad lokalnim spremištem podataka, tako i ovdje, pomoću nje se briše cijelo sesijsko spremište podataka.

Metoda key()

Slijedeći primjer biti će funkcija unutar koje će se dohvatiti ključ nekog zapisa iz sesijskog spremišta podataka koristeći indeks zapisa kao identifikator. Postupak je identičan onom iz poglavlja 4.5.1.5.Metoda key(), uz iznimku što se ovaj puta vrši nad sesijskim spremištem podataka.

```
function dohvatiZapisPrekoIndeksa() {

        sessionStorage.setItem('userId', '6576');

        var prviZapis = sessionStorage.key(0);
}
```

Kreirana je funkcija `dohvatiZapisPrekoIndeksa()` unutar koje je prvo spremljen zapis u sesijsko spremište podataka. Nakon toga je inicijalizirana varijabla 'prviZapis' je je njoj pridružena vrijednost zapisa dohvaćenog preko indeksa.

5.4.2.Primjer iz prakse - prijava

Slijedeći primjer pokazati će kako koristiti sesijsko spremište za prijavu korisnika u neku mrežnu aplikaciju. Za početak će biti prikazan javascript dio, a zatim će slijediti HTML i na kraju PHP dio gdje će biti pokazan i SQL upit kojim se pronalazi korisnik u bazi podataka. Važno je napomenuti da je ovo samo demonstracija toka podataka, te je iz tog razloga sve vrlo pojednostavljeno i sažeto

```
function login() {

        var korisnickoIme = $('#username').val();
        var lozinka = $('#password').val();
```

```
        $.ajax({
                type: "POST",
                url: "userLogin.php",
                data: {"username: korisnickoIme, password: lozinka"},
                success: function (data) {

                        sessionStorage.setItem('userId', data);
                }

        });
}

$('#btnLogin').on('click', login());
```

Ovo je pojednostavljena verzija javascript dijela u kojem ima funkcija `login()` unutar koje se inicijaliziraju dvije varijable, te se njima pridružuju vrijednosti dohvaćene iz HTML elemenata koji će kasnije biti i pokazani na HTML dijelu. Nakon toga se ajax pozivom šalju podaci u skriptu userLogin.php, te ako je skripta uspješno vratila podatak koji je potreban, radi se zapis u sesijsko spremište podataka. Na dnu je vidljivo da se funkcija poziva kada se klikne određeni gumb koji će biti pokazan u primjeru HTML-a.

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>test session storage</title>
</head>
<body>
        <div id="loginContainer">
                <input type='text' id='username' />
                <input type='password' id='password' />

                <button id='btnLogin'>Prijava</button>
        </div>
</body>
<script src="script.js" type="text/javascript"></script>
</html>
```

Ovo je primjer HTML stranice koja uz pomoć javascript dijela iz prethodnog primjera, te PHP dijela koji će u nastavku biti pokazan, vrši prijavu korisnika u neku aplikaciju. Bitno za uočiti ovdje su dva `<input/>` elementa u koja korisnik unosi korisničko ime i lozinku, te `<button>` element koji pokreće funkciju `login()`, koja je pokazana u javascript primjeru, kada se klikne na njega. Za kraj je preostao još prikaz PHP skripte.

```
<?php
require('connect.php');

$username = $_POST['username'];
$password = $_POST['password'];

$query = $db->prepare("SELECT profile_id FROM profile WHERE nick like
'".$username."' and password like '".$password."'");

$rows = $query->rowCount();

if ($rows == 1) {
```

```
        $users = array_values($query->fetch(PDO::FETCH_ASSOC));
        echo $users[0];
}
```

Ovo je primjer vrlo pojednostavljene PHP skripte u kojoj su inicijalizirane dvije varijable $username i $password, te su njima pridružene vrijednosti primljene metodom POST. Nakon toga se kreira i izvršava SQL upit, te na osnovu broja redova koji taj upit vrati se nastavlja dalje. U ovom slučaju želi se da upit pronađe samo jednog korisnika, ako je taj uvjet ispunjen, PHP skripta vraća prvi element niza $users, koji je zapravo atribut ID koji se u javascript primjeru sa početka sprema u sesijsko spremište.

6. Indeksirana baza podataka

Nakon što su obrađeni principi lokalnog spremišta podataka i sesijskog spremišta podataka, prelazi se na malo kompleksnije principe na kojima se zasnivaju indeksirane baze podataka. Kao i u poglavljima do sada, krenuti će se sa upoznavanjem sa temeljnim stvarima, kao što su, što to uopće indeksirana baza podatka je, za što se koristit, te kako se koristi. Biti će govora o programerskim sučeljima (*eng. interface)* koja stoje na raspolaganju prilikom rada sa indeksiranim bazama podataka. Kao i uvijek, ovdje neće izostati priče o sigurnosti koja je važan aspekt svake mrežne tehnologije. Jedno poglavlje će biti posvećeno događajima karakterističnim za indeksirane baze podataka, s obzirom da su osnove poput što je događaj i što je slušatelj događaja već utvrđene u poglavljima 4.4.1.Događaj i 4.4.2.Slušatelji događaja, o tome neće biti previše govora. Nakon utvrđivanja samih osnova koje su potrebne za ozbiljniju priču o indeksiranim bazama podataka, biti će pokazan možda i najbitniji princip koji je nužan prilikom rada sa indeksiranim bazama podataka, a to je sinkronizacija indeksirane baze podataka sa udaljenom bazom podataka na serveru. Za kraj će biti pokazani stvarni primjeri iz prakse, a objašnjenjima za lakše razumijevanje prikazanog sadržaja.

6.1. Što je indeksirana baza podataka

Indeksirane baze podataka su aplikacijsko programsko sučelje (eng. application programming interface, API) niske razine za pohranu značajnih količina strukturiranih podataka na strani klijenta, uključujući datoteke i velike binarne objekte (eng. Binary large object, BLOB). Ovo aplikacijsko programsko sučelje upotrebljava indekse za omogućavanje pretrage visokih performansi tih podataka. Iako su lokalna spremišta podataka korisna za pohranu manjih količina podataka, manje su korisna za pohranjivanje većih količina strukturiranih podataka. Indeksirane baze podataka pružaju rješenje.[14]

Indeksirane baze podataka su sustav koji se temelji na transakcijskim bazama podataka, poput relacijskog sustava upravljanja bazom podataka (eng. relational database management system, RDBMS) temeljenim na SQL-u. Međutim, za razliku od relacijskih sustava upravljanja bazom podataka baziranih na SQL-u, koji koriste tablice s fiksnim stupcima, indeksirane baze podataka su baze podataka temeljene na javascript skriptnom jeziku. Indeksirane baze podataka omogućuju spremanje i preuzimanje objekata koji su indeksirani ključem; mogu se pohraniti svi predmeti koji podržavaju strukturirani klon algoritam. Mora se odrediti shema baze podataka, otvoriti vezu s bazom podataka, a zatim preuzeti i ažurirati podatke u nizu transakcija.[14]

Operacije izvršene pomoću indeksiranih baza podataka izvršene su asinkrono, kako ne bi blokirale aplikacije. Indeksirane baze podataka izvorno su uključivale i sinkrona i asinkrona aplikacijsko programsko sučelje. Sinkrono aplikacijsko programsko sučelje bilo je namijenjeno samo za mrežne programere, ali je uklonjeno iz specifikacije jer nije bilo jasno da li je potrebno. Međutim, sinkrono aplikacijsko programsko sučelje može se ponovno uvesti u specifikaciju ako postoji dovoljno potražnje od mrežnih programera.[14]

6.1.1. Osnovni koncepti indeksirane baze podataka

U ovom poglavlju će se proći kroz najbitnije principe koje treba imati na umu kada se razgovara o indeksiranim bazama podataka.

- **Indeksirane baze podataka barataju sa uređenim parovima ključ/vrijednost** – vrijednosti mogu biti kompleksno strukturirani objekti, a ključevi mogu biti npr. značajke tih objekata[14]
- **Indeksirane baze podataka se zasnivaju na transakcijskim bazama podataka** – kakva god radnja se odvija prilikom korištenja indeksiranih baza podataka se odvija u kontekstu transakcije[14]
- **Indeksirane baze podataka su primarno asinkrone** – aplikacijsko programsko sučelje ne daje podatke direktnim vraćanjem vrijednosti, kao npr. poziv neke funkcije. Umjesto toga se mora postaviti funkcija povratnog poziva (*eng. callback function)* kroz koju se šalje zahtjev da se odradi neka operacija nad bazom. Nakon toga preko događaja se šalje obavijest da li je operacija uspjela ili nije. [14]
- **Indeksirane baze podataka koriste velik broj zahtjeva** – zahtjevi su objekti koji primaju uspjele ili neuspjele događaje koji su spomenuti ranije. Imaju različita svojstva poput, “onsuccess“ i “onerror“ koja čak na sebe mogu vezati i slušatelje. [14]
- **Indeksirane baze podataka koriste događaje za slanje obavijesti o ishodu zahtjeva** – događaji uvijek imaju svojstvo “type“, u ovom slučaju je ono najčešće tipa “error“ ili “success“. Također imaju i svojstvo “target“ koje označava mjesto događaja. [14]
- **Indeksirane baze podataka su objektno orijentirane** – indeksirane baze podataka nisu relacijske baze s tablicama koje predstavljaju zbirke redaka i stupaca, nego objekte određene ključevima[14]
- **Indeksirane baze podataka ne koriste SQL sintaksu** – upotrebljavaju se upiti sa indeksima koje stvara kursor, koji se pak koristi za iteraciju po skupu rezultata[14]
- **Indeksirane baze podataka se pridržavaju politike istog podrijetla** – ovdje se podrijetlo smatra domena, protokol aplikacijskog sloja i jedinstveni lokator resursa dokumenta iz kojeg se skripta izvršava. [14]

6.1.2. Programska sučelja indeksiranih baza podataka

U ovom poglavlju će samo biti nabrojana programska sučelja koja stoje na raspolaganju prilikom korištenja indeksiranih baza podataka.

- **IDBCursor** – predstavlja kursor za prelazak ili iteraciju po zapisima u bazi[14]
- **IDBCursorSync** – kursor za prelazak ili iteraciju po zapisima u bazi koji omogućuje sinkronu obradu podataka[14]
- **IDBCursorWithValue** – identičan kao IDBCursor osim što ima svojstvo “value“ [14]
- **IDBDatabase** – preko njega se ostvaruje veza sa bazom podataka[14]
- **IDBDatabaseException** – objekt koji predstavlja izuzetak prilikom rada sa bazom[14]
- **IDBDatabaseSync** – predstavlja sinkronu konekciju sa bazom[14]
- **IDBEnviroment** – sadrži svojstvo “indexedDB“ koje omogućuje pristup funkcionalnostima indeksirane baze podataka[14]
- **IDBEnviromentSync** – trenutno nije implementirano[14]
- **IDBFactory** – osigurava aplikaciji asinkroni pristup indeksiranoj bazi podataka[14]
- **IDBFactorySync** - osigurava aplikaciji sinkroni pristup indeksiranoj bazi podataka[14]
- **IDBIndex** – osigurava asinkroni pristup indeksima[14]
- **IDBIndexSync** – osigurava sinkroni pristup indeksima[14]
- **IDBKeyRange** – predstavlja kontinuirani interval s nekim vrstama podataka koji se koristi za ključeve[14]
- **IDBObjectStore** – predstavlja asinkroni pristup objektnom spremištu u bazi podataka[14]
- **IDBObjectStoreSync** - predstavlja sinkroni pristup objektnom spremištu u bazi podataka[14]
- **IDBOpenDBRequest** – osigurava pristup rezultatu zahtjeva za otvaranje ili zatvaranje baze[14]
- **IDBRequets** – omogućuje asinkroni pristup rezultatu zahtjeva prema bazi podataka koristeći upravitelje događajima[14]
- **IDBTransaction** – omogućuje statičku asinkronu transakciju prema bazi podataka koristeći upravitelje događajima[14]
- **IDBTransationSync** - omogućuje statičku sinkronu transakciju prema bazi podataka koristeći upravitelje događajima[14]

- **IDBVersionChangeEvent** – indikator koji upućuje na promjenu verzije baze podataka[14]
- **IDBVersionChangeRequest** – omogućuje slanje zahtjeva za promjenom verzije baze podataka[14]

6.2. Sigurnost indeksirane baze podataka

Kao i u prethodnim poglavljima vrijeme je da se zaviri u sigurnosni aspekt indeksiranih baza podataka. U ovom poglavlju će biti primarno govora o prijetnjama kojima je sustav izložen ukoliko koristi indeksiranu bazu podataka.

6.2.1. Dijeljenje resursa između različitih izvorišta

Dijeljenje resursa između različitih izvorišta *(eng. cross origin resource sharing, CORS)* je mehanizam koji omogućava javascript skriptnom jeziku da radi XMLHttp zahtjeve prema drugoj domeni i učitavanje podataka o odgovoru poslužitelja natrag u skriptu. Dopuštajući aplikacijama trećih strana da pristupaju podacima stvorenim od strane aplikacija drugih domena može dovesti do sigurnosnih problema, poput propuštanja informacija. CORS se proširuje na dizajn istih pravila o podrijetlu. Svaki resurs izjavljuje skup podrijetla koji mogu izdati razne vrste zahtjeva. Koristi se tehnika slijepog odgovora koju kontrolira dodatno HTTP zaglavlje, koje kada se doda zahtjevu dopušta da dosegne cilj. To znači da aplikacija stvara indeksiranu bazu podataka koja se sprema sa nazivom domene Druga aplikacija ne može pristupiti datotekama te baze podataka jer je pristup ograničen za određenu domenu. Ovaj napad se u biti temelji na zaobilaženju pravila o istom podrijetlu i uspostavljanju veza između više domena kako bi se omogućilo implementiranje zahtjeva za napad krivotvorenjem vektora (*eng. Request Forgery attack vector)*.[15]

6.2.2. Skriptiranje sa udaljenih stranica

Skriptiranje sa udaljenih stranica (*cross-site scripting, XSS)* jedan je od najpopularnijih napada na mrežnu stranicu. XSS je jako popularan napada jer iako je mrežna stranica osigurana od napada, ovaj napad se oslanja na korisnika kojeg se može zavarati da klikom na vezu odobri napad. XSS iskorištava prednost mrežnih aplikacija kod kojih korisnički unos nije pravilno filtriran. Napadači također mogu manipulirati neizravnim ulazima kao što su varijable sesije i zapisi baze podataka. XSS je tehnika napada koja prisiljava mrežnu stranicu da prikaže zlonamjeran kod, koji se izvršava u korisničkom mrežnom pregledniku. S obzirom da indeksirane baze podataka omogućavaju pohranu podataka na strani korisnika, a ti pohranjeni podaci mogu sadržavati osjetljive podatke, kao što su osobni podaci korisnika. Ukoliko je

mrežna aplikacija ranjiva na XSS napad, napadač može pristupiti pohrani na strani klijenta. [15]

6.2.3. Napadi društvenog inženjerstva

Društveno inženjerstvo je umijeće manipulacije ljudima sa ciljem da se otkriju zaštićene podatke. Napadači obično varaju ljude tako da saznaju od njih lozinke ili bankovne informacije ili da im daju pristup računalima da bi im u tajnosti postavili maliciozne programe sa ciljem izvlačenja informacija ili kontrole računala. Napadači koriste ovaj oblik napada jer je lakše iskoristiti lakovjernu ljudsku prirodu nego zaobići zaštite na nekoj aplikaciji. Primjer društvenog inženjerstva može biti elektronička poruka od prijatelja. Ako napadač uspije na bilo koji način upasti u korisnikov račun elektroničke pošte, automatski će imati pristup svim njegovim kontaktima. Tako napadač može poslati zlonamjeran sadržaj svim kontaktima žrtve u svrhu preuzimanja računala ili izvlačenja osjetljivih informacija. Na taj način indeksirana baza podataka može biti ugrožena. [15]

6.2.4. Fizički pristup

Fizički pristup je moguć kada napadač dođe u fizičko prisustvo korisničkog računala. Ako uređaj ili korisnički podaci nisu kriptirani napadač može dobiti pristup svim podacima. Ovaj problem je moguće izbjeći kontrolom fizičkog pristupa prostorijama gdje se drže računala, ako se radi o poduzeću. U svakom slučaju korisnik bi uvijek trebao zaključavati zaslon koji je zaštićen lozinkom kada je odsutan od računala. [15]

6.3. Događaji indeksirane baze podataka

Ovo poglavlje će biti posvećeno događajima koji stoje na raspolaganju prilikom korištenja indeksiranih baza podataka. S obzirom da je u ranijim poglavljima utvrđeno što je događaj i što su slušatelji događaj, o tome neće biti govora u ovom poglavlju koje slijedi. Za razliku od sesijskog spremišta podataka i lokalnog spremišta podataka koji zapravo raspolažu samo jednim događajem, indeksirane baze podataka raspolažu sa čak 8 različitih događaja. To su upgradeneeded, complete, abort, success, error, blocked, versionchange i close. Svaki od njih će biti objašnjem i potkrepljen primjerom.

6.3.1. Događaj upgradeneeded

Događaj upgradeneeded se pokreće kada se učita baza podataka čija je verzija veća od trenutne verzije pohranjene baze podataka. Upravitelj događaja koji se koristi prilikom događaja upgradeneeded je dio sučelja IDBOpenDBRequest, a naziva se onupgradeneeded.

Događaj koji se proslijeđuje slušatelju je sučelje IDBVersionChangeEvent. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća:

```
IDBOpenDBRequest.onupgradeneeded = function(event) { //neka akcija };
```

Slijedeći primjer će pokazati kako koristiti onupgradeneded upravitelj koji se koristi za ažuriranje strukture baze podataka ako se učita baza podataka s višim brojem verzije.[14]

```
var db;

var request = window.indexedDB.open("library", 3);

request.onupgradeneeded = function(event) {
  db = request.result;

  db.onerror = function(errorEvent) {
    note.innerHTML += '<li>Error loading database.</li>';
  };

  if (event.oldVersion < 1) {
    var store = db.createObjectStore("books", {keyPath: "isbn"});
    var titleIndex = store.createIndex("by_title", "title", {unique:
true});
    var authorIndex = store.createIndex("by_author", "author");
  }
  if (event.oldVersion < 2) {
    var bookStore = request.transaction.objectStore("books");
    var yearIndex = bookStore.createIndex("by_year", "year");
  }
  if (event.oldVersion < 3) {
    var magazines = db.createObjectStore("magazines");
    var publisherIndex = magazines.createIndex("by_publisher",
"publisher");
    var frequencyIndex = magazines.createIndex("by_frequency",
"frequency");
  }
};

request.onerror = function(event) {
  note.innerHTML += '<li>Error loading database.</li>';
};

request.onsuccess = function(event) {
  note.innerHTML += '<li>Database initialised.</li>';
  db = request.result;
  populateAndDisplayData();
};
```

U prikazanom primjeru funkcija treba stvoriti novu verziju baze podataka. Verzija 1 je prva verzija baze podataka. Sa verzijom 2 je predstavljen novi indeks knjiga prema godinama. Sa verzijom 3 je predstavljeno novo objektno spremište za časopise sa dva indeksa. [14]

6.3.2. Događaj complete

Događaj complete se pokreće u trenutku kada je neka transakcija uspješno obavljena. Upravitelj događaja koji se koristi prilikom događaja complete je dio sučelja IDBTransaction, a naziva se oncomplete. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća.

```
transaction.oncomplete = function(error) { //neka akcija };
```

U slijedećem primjeru koda otvoriti će se transakcija za čitanje i pisanje u bazi podataka, te će se dodati neki podaci u objektno spremište. Treba imati na umu da funkcije koje su pridružene transakcijskim upraviteljima događaja će izvijestiti o ishodu transakcije bila ona uspješna ili ne. [14]

```
var DBOpenRequest = window.indexedDB.open("toDoList", 4);

DBOpenRequest.onsuccess = function(event) {
  note.innerHTML += '<li>Database initialised.</li>';

  db = DBOpenRequest.result;

  addData();
};

function addData() {
  var newItem = [ { taskTitle: "Walk dog", hours: 19, minutes: 30, day: 24,
month: "December", year: 2013, notified: "no" } ];

  var transaction = db.transaction(["toDoList"], "readwrite");

  transaction.oncomplete = function(event) {
    note.innerHTML += '<li>Transaction completed: database modification
finished.</li>';
  };

  transaction.onerror = function(event) {
    note.innerHTML += '<li>Transaction not opened due to error: ' +
transaction.error + '</li>';
  };

  var objectStore = transaction.objectStore("toDoList");

  var objectStoreRequest = objectStore.add(newItem[0]);

  objectStoreRequest.onsuccess = function(event) {
    note.innerHTML += '<li>Request successful.</li>';
  };
};
```

U ovom primjeru za početak se otvara baza podataka. Nakon toga se sprema rezultat otvaranja baze podataka u varijablu 'db'. Poziva se funkcija `addData()` da se dodaju podaci u bazu podataka. Funkcija `addData()` kreira novi objekt spreman za umetanje u bazu podataka. Nakon toga otvara se čitaj/piši transakcija prema bazi podataka spremna za umetanje podataka. Vraća se notifikacija da je transakcija bila uspješna ili neuspješna. Kreira se objektno

spremište na transakciji. Dodaje se novi objekt u objektno spremište. Vraća se notifikacija da je zahtjev uspješan. Važno je napomenuti da to ne znači da je podatak uspješno završio u bazi podataka. Za to bi trebalo koristiti `transaction.onsuccess`. [14]

6.3.3. Događaj abort

Događaj abort se pokreće u trenutku kada je neka transakcija opozvana. Upravitelj događaja koji se koristi prilikom događaja abort je dio sučelja IDBTransaction, a naziva se onabort. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća.

```
transaction.onabort = function(error) { ... };
```

U slijedećem primjeru koda otvoriti će se transakcija za čitanje i zapisivanje u bazu podataka, te će se dodati neki podaci u objektno spremište. Kao i u prethodnom primjeru, treba imati na umu da funkcije koje su pridružene transakcijskim upraviteljima događaja će izvijestiti o ishodu transakcije bila ona uspješna ili ne. Za kraj valja napomenuti da funkcija onabort se pokreće samo kada je transakcija opozvana. [14]

```
var DBOpenRequest = window.indexedDB.open("toDoList", 4);

DBOpenRequest.onsuccess = function(event) {
  note.innerHTML += '<li>Database initialised.</li>';

  db = DBOpenRequest.result;

  addData();
};

function addData() {
  var newItem = [ { taskTitle: "Walk dog", hours: 19, minutes: 30, day: 24,
month: "December", year: 2013, notified: "no" } ];

  var transaction = db.transaction(["toDoList"], "readwrite");

  transaction.oncomplete = function(event) {
    note.innerHTML += '<li>Transaction completed: database modification
finished.</li>';
  };

  transaction.onerror = function(event) {
    note.innerHTML += '<li>Transaction not opened due to error: ' +
transaction.error + '</li>';
  };

  var objectStore = transaction.objectStore("toDoList");

  var objectStoreRequest = objectStore.add(newItem[0]);

  objectStoreRequest.onsuccess = function(event) {
    note.innerHTML += '<li>Request successful.</li>';
  };

  transaction.onabort = function() {
    console.log("Transaction aborted!");
```

```
  };

  transaction.abort();
};
```

Početak ovog primjera je identičan kao prošli primjer. Jedina razlika se javlja na samom kraju funkcije `addData()`. S obzirom da je tok objašnjen u prošlom poglavlju, krenuti će se od `transaction.onabort`. Pozvan je upravitelj događaja onabort koji će odraditi akciju u trenutku kada je transakcija uspješno opozvana. Nakon toga se opoziva transakcija. [14]

6.3.4. Događaj success

Događaj success se pokreće u trenutku kada je rezultat nekog zahtjeva uspješan. Upravitelj događaja koji se koristi prilikom događaja success je dio sučelja IDBRequest, a naziva se onsuccess. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća.

```
request.onsuccess = function(event) { ... };
```

Slijedeći primjer koda će biti zahtjev kojim se pokušava dobiti zadani naslov zapisa. Funkcija događaja onsuccess će dobiti pridruženi zapis iz sučelja IDBObjectStore, koji je dostupan kao objectStoreTitleRequest.result. Ažurira se jedan entitet zapisa, te se onda sprema ažurirani zapis u objektno spremište. [14]

```
var title = "Walk dog";

var objectStore = db.transaction(['toDoList'],
"readwrite").objectStore('toDoList');

var objectStoreTitleRequest = objectStore.get(title);

objectStoreTitleRequest.onsuccess = function() {
  var data = objectStoreTitleRequest.result;

  data.notified = "yes";

  var updateTitleRequest = objectStore.put(data);

  updateTitleRequest.onsuccess = function() {
    displayData();
  };
};
```

U primjeru se inicijalizira varijabla title i pridružuje joj se vrijednost. Otvara se transakcija kao i u primjerima do sada. Dohvaća se objekt koji ima naziv jednak vrijednosti spremljenoj unutar varijable 'title'. Hvata se podatkovni objekt koji je uspješno vraćen kao rezultat. Ažurira se vrijednost objekta `data.notified` u vrijednost 'yes'. Kreira se još jedan zahtjev koji će spremiti objekt nazad u bazu podataka. Kada taj novi zahtjev prođe uspješno pokrenut će se funkcija displayData().[14]

6.3.5. Događaj error

Događaj error se pokreće u trenutku kada se za vrijeme transakcije dogodi nepredviđena greška. Upravitelj događaja koji se koristi prilikom događaja error je dio sučelja IDBTransaction, a naziva se onerror. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća.

```
transaction.onerror = function(event) { ... };
```

U slijedećem primjeru koda otvoriti će se transakcija za čitanje i zapisivanje u bazu podataka, te će se dodati neki podaci u objektno spremište. Kao i u prva dva primjera, treba imati na umu da funkcije koje su pridružene transakcijskim upraviteljima događaja će izvijestiti o ishodu transakcije bila ona uspješna ili ne. Za kraj valja napomenuti da se funkcija onerror pokreće samo kada je transakcija neuspješna i prijavljuje do koje je greške došlo. [14]

```
var DBOpenRequest = window.indexedDB.open("toDoList", 4);

DBOpenRequest.onsuccess = function(event) {
  note.innerHTML += '<li>Database initialised.</li>';

  db = DBOpenRequest.result;

  addData();
};

function addData() {
  var newItem = [ { taskTitle: "Walk dog", hours: 19, minutes: 30, day: 24,
month: "December", year: 2013, notified: "no" } ];

  var transaction = db.transaction(["toDoList"], "readwrite");

  transaction.oncomplete = function(event) {
    note.innerHTML += '<li>Transaction completed: database modification
finished.</li>';
  };


  transaction.onerror = function(event) {
    note.innerHTML += '<li>Transaction not opened due to error: ' +
transaction.error + '</li>';
  };

  var objectStore = transaction.objectStore("toDoList");

  var objectStoreRequest = objectStore.add(newItem[0]);

  objectStoreRequest.onsuccess = function(event) {
    note.innerHTML += '<li>Request successful.</li>';
  };
};
```

S obzirom da je cijeli kod izuzev jedne linije identičan primjeru iz poglavlja 6.5.2.Događaj complete, nema potrebe da se ponovno sve objašnjava. Objasniti će se jedino nova linija:

```
transaction.onerror = function(event) {
```

```
    note.innerHTML += '<li>Transaction not opened due to error: ' +
transaction.error + '</li>';
  };
```

Kao što je vidljivo definirana je funkcija koja unutar sebe sadrži akciju koja će biti odrađena ukoliko transakcija bude neuspješna. Također će greška do koje je došlo biti ispisana. [14]

6.3.6. Događaj blocked

Događaj blocked se pokreće u trenutku kada bi trebao biti odrađen događaj upgradeneeded zbog izmjene na verziji baze, ali baza se i dalje koristi. Upravitelj događaja koji se koristi prilikom događaja blocked je dio sučelja IDBOpenDBRequest, a naziva se onblocked. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća.

```
IDBOpenDBRequest.onblocked = function(event) { ... };
```

Slijedeći primjer će pokazati kako koristiti onblocked upravitelj koji se koristi za blokiranje ažuriranja strukture baze podataka ako se baza podataka još uvijek koristi. [14]

```
var db;

var request = indexedDB.open("toDoList", 4);

request.onerror = function(event) {
  note.innerHTML += '<li>Error loading database.</li>';
};

request.onsuccess = function(event) {
  note.innerHTML += '<li>Database initialised.</li>';

  db = request.result;

  displayData();
};

request.onupgradeneeded = function(event) {
  var db = event.target.result;

  db.onerror = function(event) {
    note.innerHTML += '<li>Error loading database.</li>';
  };

  var objectStore = db.createObjectStore("toDoList", { keyPath: "taskTitle"
});

};

request.onblocked = function() {
  console.log("Your database version can't be upgraded because the app is
open somewhere else.");
}
```

U prikazanom primjeru otvara se baza podataka. Priključuju se upravitelji događaja za uspješno i za neuspješno otvaranje baze podataka. Sprema se rezultat otvaranja baze podataka u varijablu 'db'. Pokreće se funkcija `displayData()`. Idući događaj se brini o tome

treba li baza preći na novu verziju. Nakon toga se kreira objektno spremište za bazu podataka. Na kraju je priključen upravitelj događaja koji odrađuje akciju, u ovom slučaju samo javlja, da nije moguće preći na novu verziju baze podataka. [14]

6.3.7. Događaj versionchange

Događaj versionchange se pokreće kada se promjeni struktura baze podataka. Upravitelj događaja koji se koristi prilikom događaja versionchange je dio sučelja IDBDatabase, a naziva se onversionchange. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća:

```
IDBDatabase.onversionchange = function(error) { ... }
```

Slijedeći primjer koda prikazuje stvaranje novog objektnog spremišta. Također su uključene funkcije onabort i onerror za obradu neuspjelih slučajeva i naravno onversionchange da obavijesti da je došlo do izmjene strukture baze podataka. [14]

```
request.onupgradeneeded = function(event) {
  var db = event.target.result;

  db.onerror = function(event) {
    note.innerHTML += '<li>Error opening database.</li>';
  };

  db.onabort = function(event) {
    note.innerHTML += '<li>Database opening aborted!</li>';
  };

  var objectStore = db.createObjectStore("toDoList", { keyPath: "taskTitle"
});

  objectStore.createIndex("hours", "hours", { unique: false });
  objectStore.createIndex("minutes", "minutes", { unique: false });
  objectStore.createIndex("day", "day", { unique: false });
  objectStore.createIndex("month", "month", { unique: false });
  objectStore.createIndex("year", "year", { unique: false });

  objectStore.createIndex("notified", "notified", { unique: false });

  note.innerHTML += '<li>Object store created.</li>';

  db.onversionchange = function(event) {
    note.innerHTML += '<li>a database change has occurred; you should
refresh this
                      browser window, or close it down and use the other
open version of
                      this application, wherever it exists.</li>';
  };
};
```

U prikazanom primjeru inicijalizira se funkcija unutar upravitelja događaja priključenog na zahtjev. Unutar funkcije se priključuju upravitelji događaja koji brinu o neuspjelim slučajevima. Kreira se novo objektno spremište, te se definiraju objekti koje će to objektno spremište

sadržavati. Na kraju se priključuje upravitelj onversionchange, te se postavlja poruka koja će biti ispisana ukoliko dođe do izmjene strukture baze podataka. [14]

6.3.8. Događaj close

Događaj close se pokreće kada se veza s bazom nepredviđeno prekine. To može biti uslijed nenadanog gašenja aplikacije ili prekida konekcije sa diskom gdje se nalazi baza podataka. Upravitelj događaja koji se koristi prilikom događaja close je dio sučelja IDBDatabase, a naziva se onclose. Sintaksa kojom se inicijalizira poziv ovog događaja je slijedeća:

```
IDBDatabase.onclose = function(error) { ... };
```

Slijedeći primjer će pokazati funkciju koja će biti pozvana ukoliko se veza sa bazom podataka neočekivano prekine. [14]

```
db.onclose = function(event) {

  myAppShowAlert('The database "' + db.name + '" has unexpectedly
closed.');

};
```

U prikazanom primjeru ukoliko dođe do neočekivanog prekida komunikacije sa bazom podataka sustav će izbaciti upozorenje da je baza podataka nenadano zatvorena. [14]

7. Aplikacija eGlas

U ovom poglavlju biti će govora o aplikaciji unutar koje su definirana sva spomenuta načela koja se tiču lokalnih spremišta podataka. Na primjerima će biti demonstrirano kako aplikacija radi.

7.1. Općenito o aplikaciji

Prilikom izrade aplikacije tehnologije koje su korištene su: PHP programski jezik na serveru za komunikaciju sa bazom podataka preko SQL upita. Vrsta baze podataka je MySql. Na klijentskoj strani je korišten javascript, HTML i CSS, a od programskih okvira jQuery i Bootstrap. Struktura same aplikacije je jednostranična aplikacija *(eng. Single Page Application – SPA*).

Glavna ideja aplikacije je generička izrada zasebnih pitanja ili anketa u cijelosti. Aplikacija zahtijevata prijavu, tj. prvotno registraciju. Iako se koristi PHP ne koristi se sesiju za provjeru valjanosti korisničke sesije. Za tu svrhu se koristi sesijsko spremište podataka.

Prilikom izrade zasebnog pitanja korisnik na raspolaganju ima tri tipa odgovora koje je u mogućnosti postaviti kao odgovor na neko pitanje. Prvi tip odgovora je biti radio gumb (*eng. Radio Button*) sa mogućnosti dodavanja neograničenog broja opcija. Drugi tip odgovora je gumb za višestruki izbor (*eng. Checkbox Button*) sa mogućnosti dodavanja neograničenog broja opcija. Treći tip odgovora je kvantitativni numerički klizač (*eng. Range Slider*) sa proizvoljno odredljivim elementima, minimalna vrijednost, maksimalna vrijednost i veličina koraka.

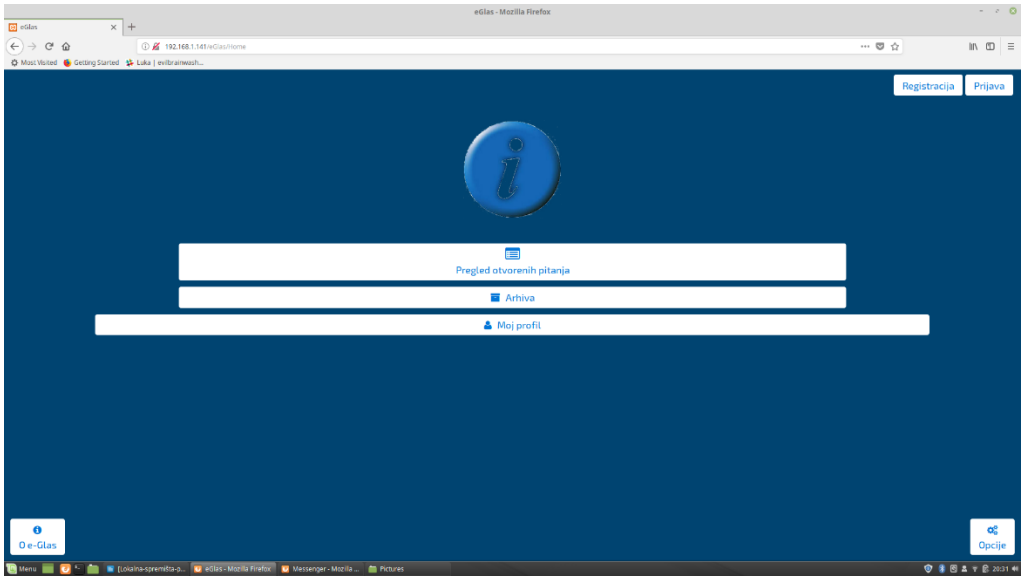
Prilikom izrade ankete moguće je dodati kratki opis ankete koji je prikazan na listi anketa. Moguće je u jednu anketu dodati proizvoljan broj pitanja za koja vrijede ista pravila kao i prilikom kreiranja zasebnog pitanja.

Osim kreiranja i objave zasebnog pitanja i ankete, svaki korisnik ima uvid u ostala objavljena zasebna pitanja i ankete, kao i u arhivu zasebnih pitanja i anketa. Moguće je odgovarati na tuđa zasebna pitanja i ankete.

Nakon isteka zasebnog pitanja ili ankete korisnik je u mogućnosti pregledati rezultate odgovora na vlastito zasebno pitanje ili anketu. Rezultati su prikazani grafički.

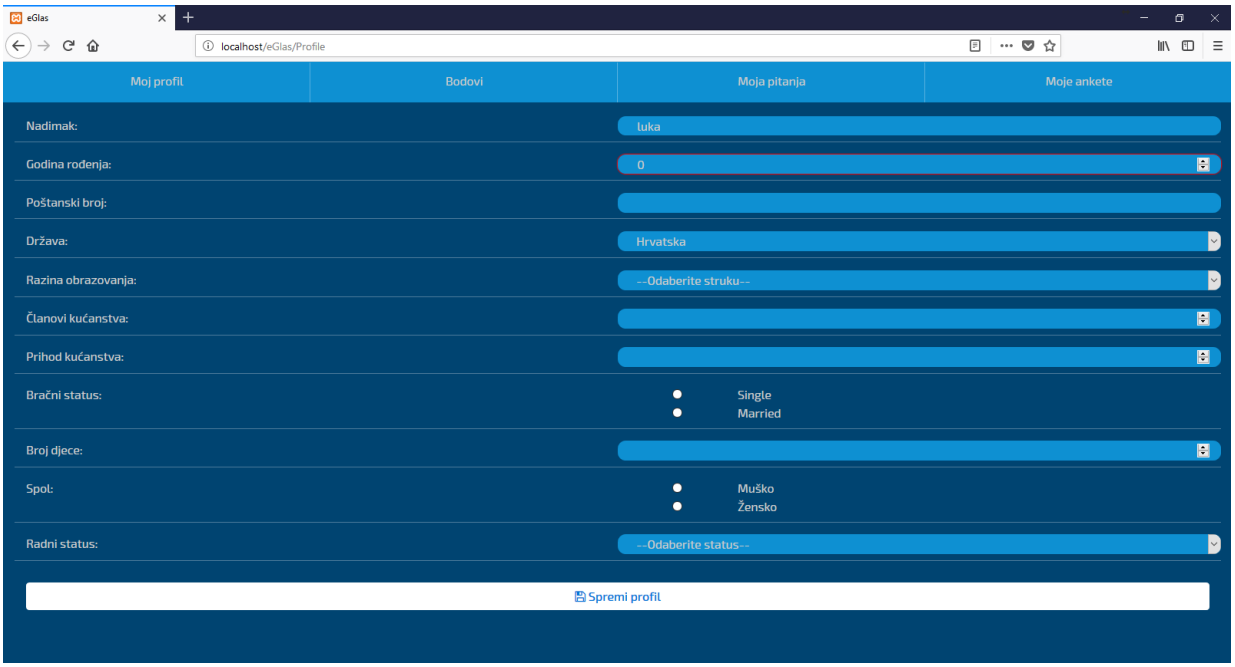
7.2. Korisničko sučelje

U ovom poglavlju će biti prikazano korisničko sučelje sa primjerima iz same aplikacije i uputama za korištenje. Započeti će se sa prvim ekranom koji se učita kada korisnik kreće sa korištenjem aplikacije.



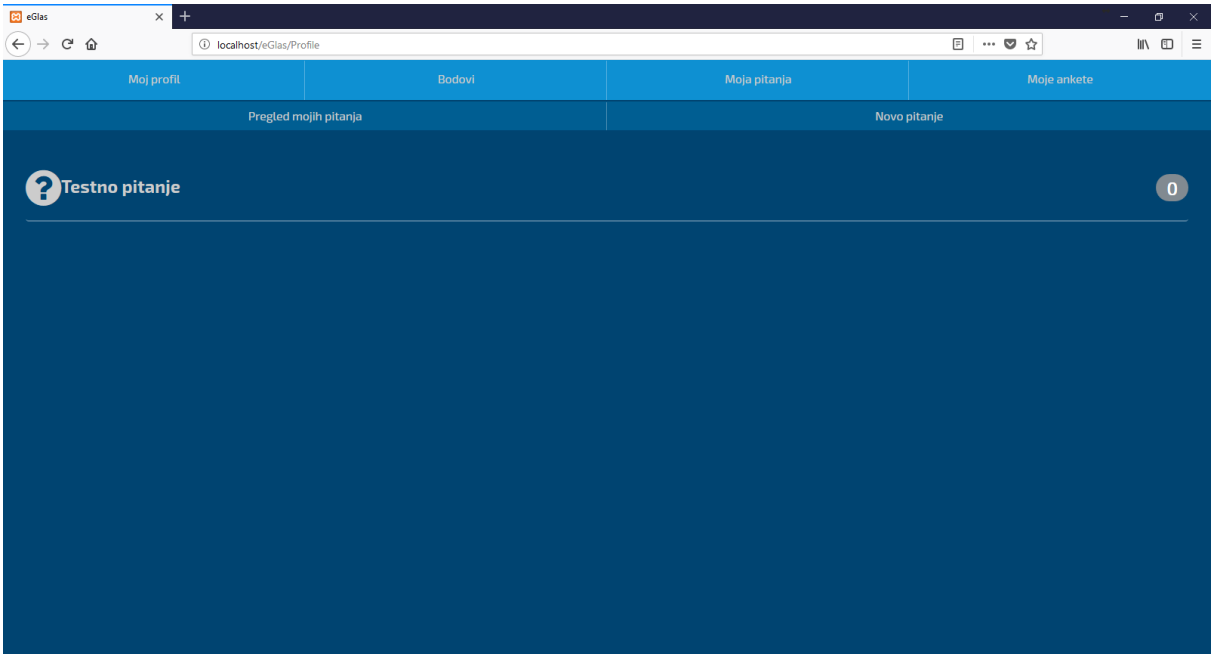
Slika 4: Prikaz početne stranice

Na prikazu Slika 4 je vidljiva navigacija. U gornjem desnom kutu stoje gumbi za registraciju i prijavu korisnika. Pregled anketa i pitanja se otvara klikom na gumb pregled otvorenih pitanja. Klikom na gumb arhiva otvara se arhiva anketa i pitanja. U donjem desnom kutu se nalazi gumb za opcije, a donji lijevi kut sadrži gumb u kojemu se može saznati nešto o aplikaciji. Gumb moj profil otvara profil korisnika unutar kojeg je moguće izmijeniti profil, kreirati ankete i pitanja, te pregledavati ankete i pitanja.



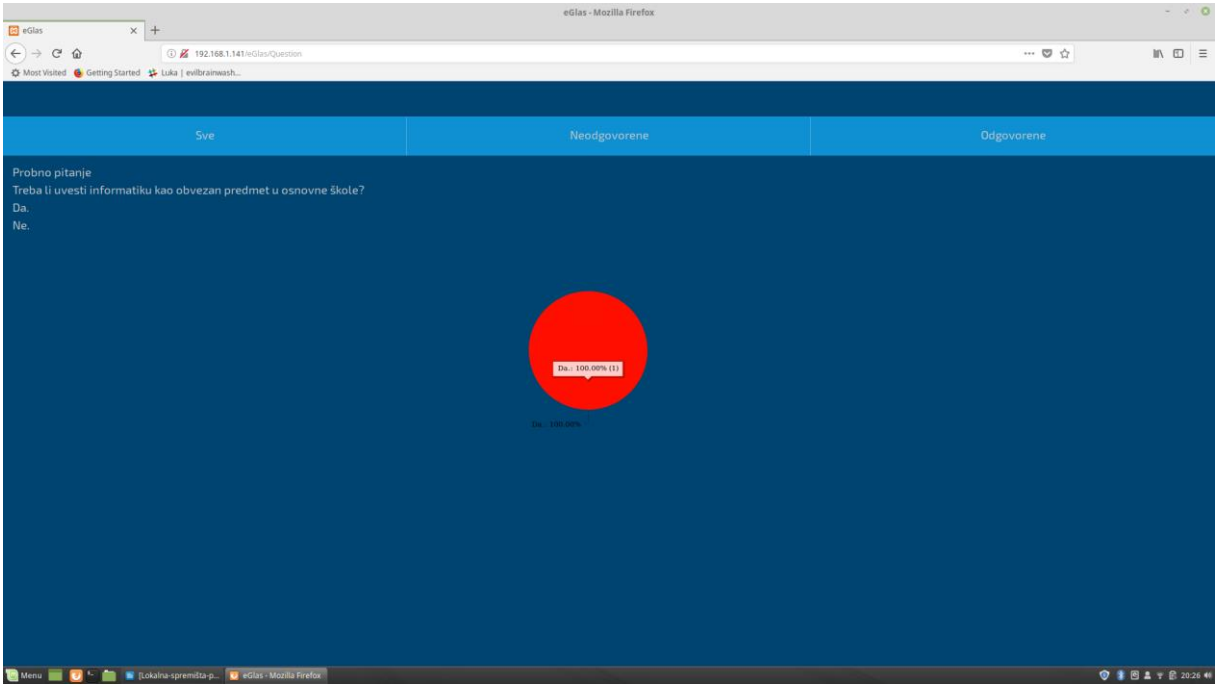
Slika 5: Prikaz profilne stranice

Na prikazu Slika 5 vidljiva je profilna stranica na kojoj je moguće izmijeniti podatke o korisniku, te na vrhu postoji navigacija koja vodi do drugih segmenata profilne stranice. Bodovi, pitanja i ankete.



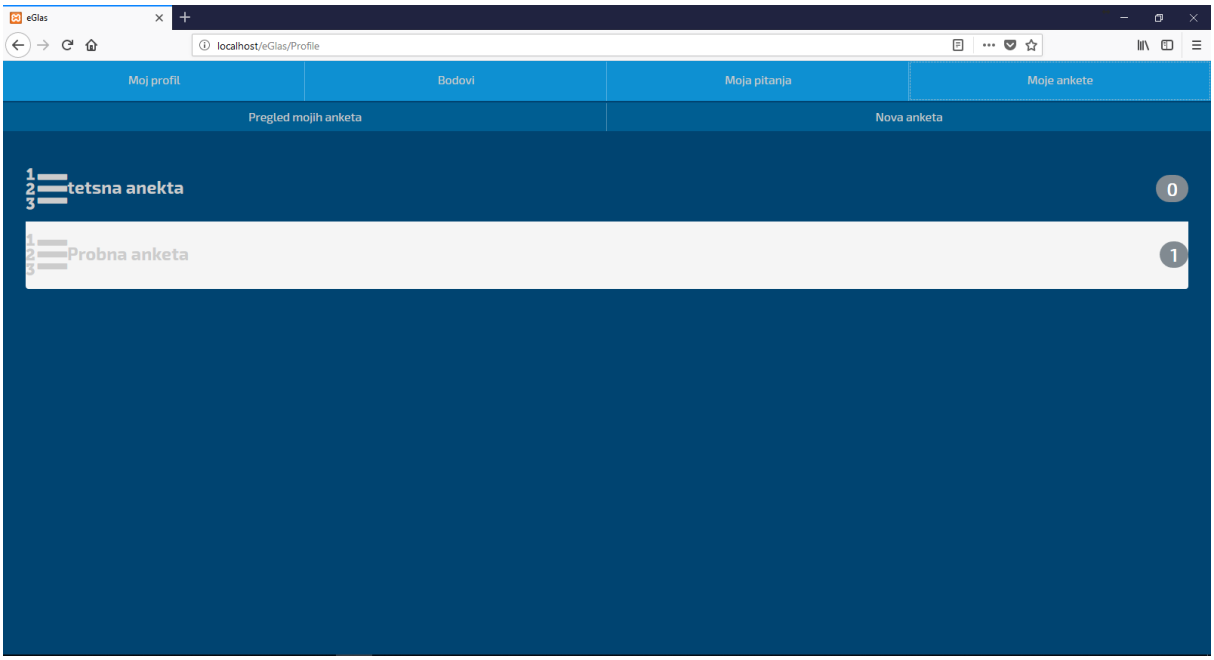
Slika 6: Prikaz liste korisničkih pitanja

Na prikazu Slika 6 su vidljiva pitanja kreirana od strane prijavljenog korisnika. Klikom na pitanje korisnik je u mogućnosti vidjeti rezultate. Klikom na novo pitanje korisnik je u mogućnosti kreirati novo pitanje.



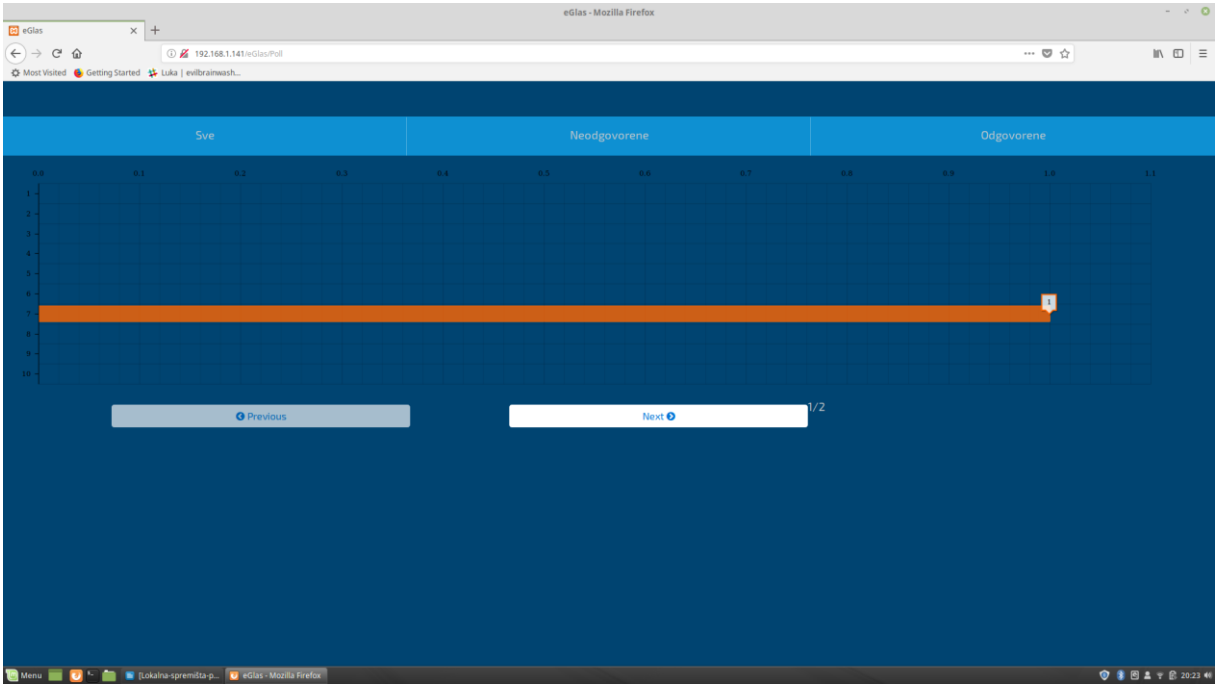
Slika 7: Prikaz rezultata korisničkih pitanja

Na prikazu Slika 7 vidljiv je grafički prikaz rezultata odgovora na pitanje, kao i samo pitanje.



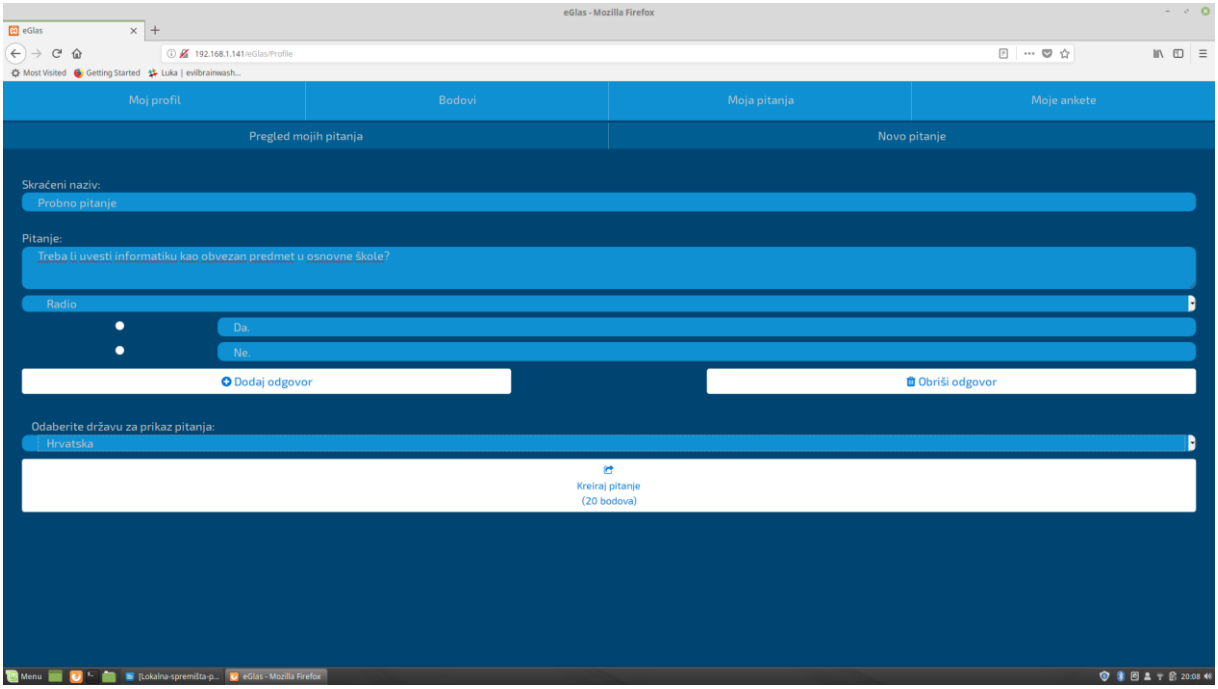
Slika 8: Prikaz rezultata korisničkih pitanja

Na prikazu Slika 8 vidljiv je prikaz liste anketa koje je korisnik kreirao. Klikom na anketu korisnik može vidjeti grafički prikaz rezultata. Klikom na nova anketa korisnik je u mogućnosti kreirati novu anketu.



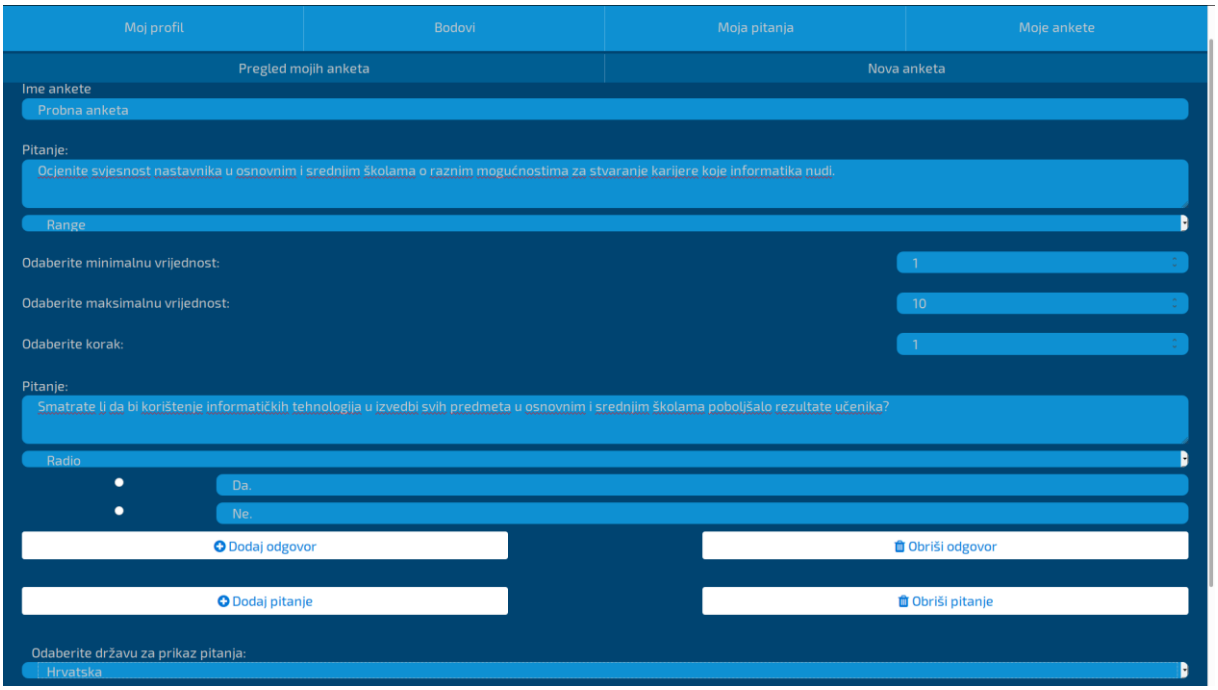
Slika 9: Prikaz rezultata korisničkih anketa

Na prikazu Slika 9 su vidljivi rezultati u obliku grafičkog prikaza odgovora na neku anketu.



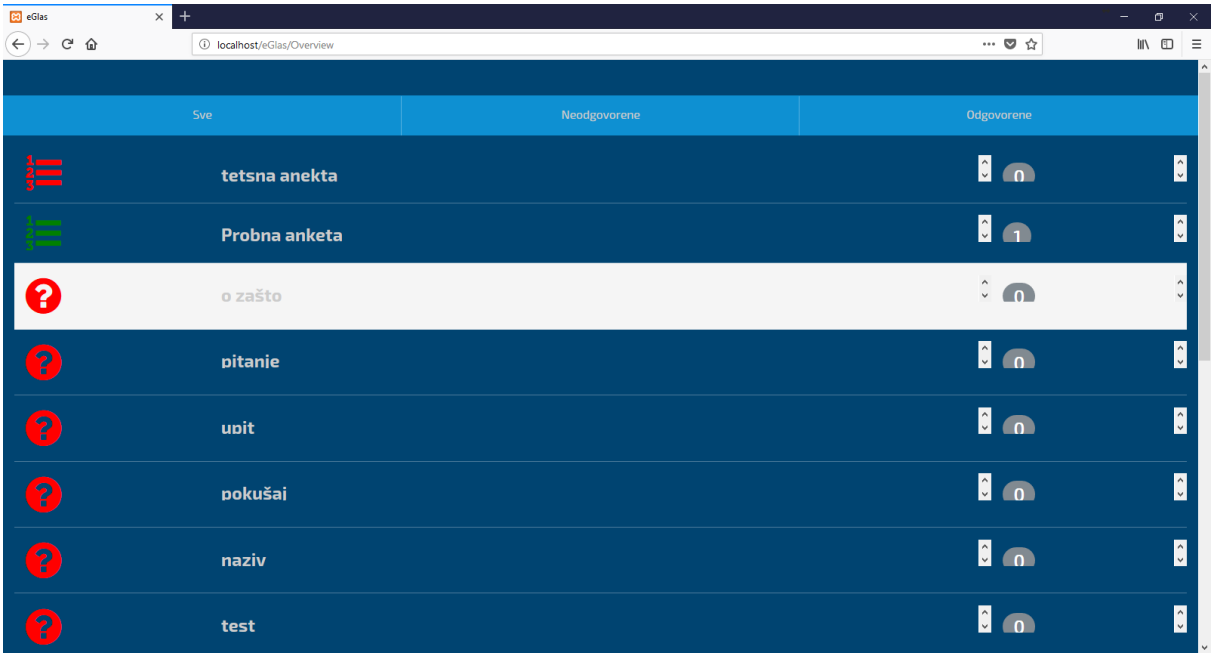
Slika 10: Prikaz kreiranja pitanja

Na primjeru Slika 10 je prikazano kreiranje pitanja. Klikom na dodaj odgovor se niže još dodatnih odgovora, a vrsta odgovora se može izabrati u polju iznad. Klikom na Kreiraj pitanje se završava postupak i pitanje se kreira.



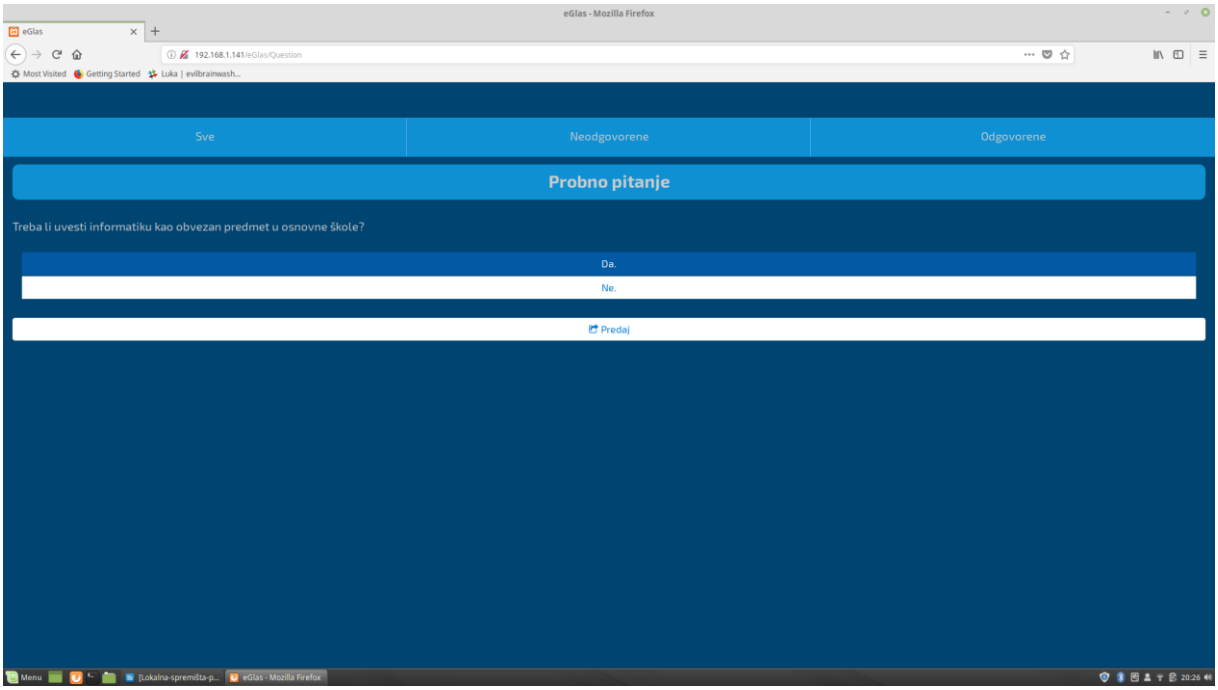
Slika 11: Prikaz kreiranja ankete

Na primjeru Slika 11 je prikazan ekran kreiranja ankete. Anketa se kreira na isti način kao i pitanje, uz iznimku što se radi o većem broju pitanja.



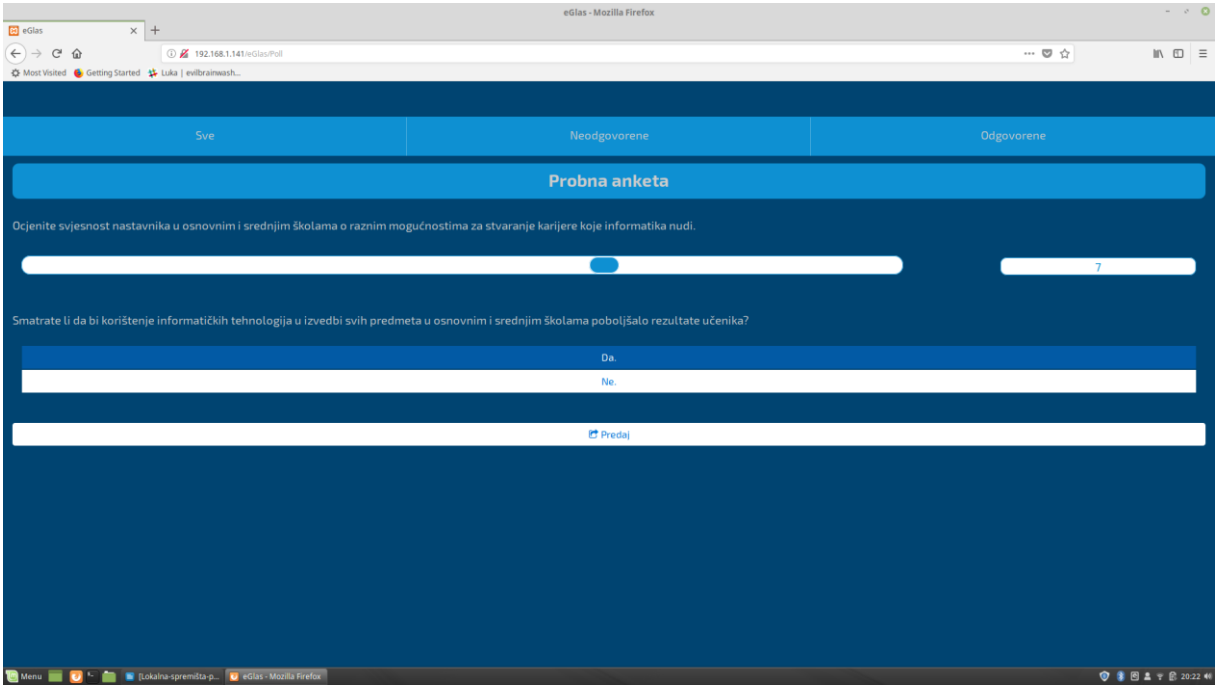
Slika 12: Prikaz liste anketa i pitanja

Na primjeru Slika 12 je prikazan ekran gdje je vidljiva lista svih pitanja i anketa unutar aplikacije kreiranih od strane drugih korisnika. Također su vidljivi filteri na vrhu za pregled samo neodgovorenih, svih ili samo odgovorenih anketa i pitanja.



Slika 13: Prikaz odgovaranja na pitanje

Na primjeru Slika 13 je pokazano kako izgleda ekran gdje se odgovara na korisničko pitanje.



Slika 14: Prikaz odgovaranja na anketu

Na primjeru Slika 14 pokazno je kako izgleda ekran gdje se odgovara na anketu korisnika.

7.3. Programski kod

U ovom, završnom poglavlju još je preostalo prikazati programski kod na nekolicini primjera iz aplikacije. primarno će se krenuti sa HTML dokumentom. S obzirom da se ovdje radi o jednostraničnoj aplikaciji, postoji samo jedan HTML dokument koji će biti prikazan u nastavku. Za početak slijedi prikaz početne stranice sa navigacijom.

```
<div id="home_screen" class="">

        <span id="uname_header" class="pull-left d-none"></span>
        <button id="about_navigate" style="position: absolute;bottom:
12px;left:12px;" class="btn btn-primary pull-left"><i class="fa fa-info-
circle" aria-hidden="true"></i></br>O e-Glas
        </button>

        <button id="login_btn" class="btn btn-primary pull-right"
style="margin-left: 5px" data-toggle="modal" data-
target="#loginModal">Prijava</button>
        <button id="register_btn" class="btn btn-primary pull-right" data-
toggle="modal" data-target="#registerModal">Registracija</button>
        <button id="logout_btn" class="btn btn-primary pull-right d-
none">Odjava</button>

        <div style="clear: both;" class="row text-xs-center"><img
style="margin-top: 5vh; max-height: 20vh;" src="images/info.png"></div>

        <button style="margin-top: 5vh;" class="mar-top-12 btn btn-primary
col-xs-12 col-md-8 col-md-offset-2" id="overview_navigate"><i style="font-
size: 28px;" class="fa fa-list-alt" aria-hidden="true"></i><br>Pregled
otvorenih pitanja</button>

        <button class="mar-top-12 btn btn-primary col-xs-8 col-xs-offset-2"
id="archive_navigate"><i style="margin-right:10px" class="fa fa-archive"
aria-hidden="true"></i>Arhiva</button>

        <button class="mar-top-12 btn btn-primary col-xs-10 col-xs-offset-
1" id="profile_navigate" ><i style="margin-right:10px" class="fa fa-user"
aria-hidden="true"></i>Moj profil</button>

    </div>
```

U ovom primjeru je vidljivo da je zapravo cijela početna stranica sačinjena od gumbi koji imaju različite akcije koje su opisane u javascript dijelu, slijedi prikaz javascript dijela prikazanih gumbi sa početne stranice.

```
$('#about_navigate').click(function () {

        history.pushState('About', "", "/eGlas/About");
        localStorage.setItem('currentPage',"about");

        $('#about').show();
        $('#home_screen').hide();
    });
```

Ovo je prikaz akcije koja se odrađuje pritiskom na gumb “O e-Glas“. Kao što je vidljivo kretanje se odrađuje tako da se samo mijenja vidljivost elemenata iz HTML dokumenta. Također se sprema u lokalno spremište trenutna lokacija koja je neophodna za funkcioniranje sistemskog gumba povratak na željeni način.

```
    $('#overview_navigate').click(function () {

        var $user_id = sessionStorage.getItem('user');

        if ($user_id==null) {
            authentication_failed();
        } else {

            history.pushState('Overview', "", "/eGlas/Overview");
            localStorage.setItem('currentPage','overview');

            $.ajax({
                type: "POST",
                url: "php/display_all.php",
                data: {id: $user_id},
                success: function (data) {
                    $("#overview_display").html(data);
                }
            });

            $('#profile_back_overview').addClass('profile_back');
            $('#overview').show();
            $('#home_screen').hide();
            $('#overview_display').show();
            $('#overview_nav').show();
            $('#details_display').hide();
        }
    });
```

U ovom primjeru je prikazana funkcija koja se poziva pritiskom na gumb “ Pregled otvorenih pitanja“. Provjerava se valjanost korisničkih podataka, radi se zapis trenutne lokacije u lokalno spremište, te se radi ajax poziv prema PHP skripti gdje se odrađuje dio koda vezan za povlačenje podataka iz baze podataka. Na samom kraju se vrši izmjena vidljivosti HTML elementima. S obzirom da tema ovog rada nije PHP neće se ići u detalje kako i zašto radi server dio aplikacije. Dovoljno je znati da nizom upita na bazu PHP skripta vraća podatke koji su potrebni. U ovom slučaju se radi o listi svih pitanja i anketa.

```
  $('#archive_navigate').click(function () {

        var $user = sessionStorage.getItem('user');

        if ($user==null) {
            authentication_failed();
        } else {

            history.pushState('Archive', "", "/eGlas/Archive");
            localStorage.setItem('currentPage',"archive");

            $.ajax({
                type: 'POST',
```

```
                url: 'php/arhiva_display.php',
                data: {user: $user},
                success: function(data){

                $('#archive_overview').html(data);

                }
            });

            $('#profile_back_archive').addClass('back_archive');
            $('#archive').show();
            $('#home_screen').hide();
        }
    });
```

U ovom primjeru je prikazana funkcija koja se poziva pritiskom na gumb “Arhiva“. Kao što je vidljivo, funkcija je jako slična prethodnoj, jedina je razlika u podacima koji se zapisuju u lokalno spremište, elementima koji se prikazuju, te setu podataka koji vraća PHP skripta. ovdje se radi o listi pitanja i anketa kojima je isteklo trajanje.

Gumb “Moj profil“ nije potrebno prikazivati jer u načelu radi jednako kao prethodna dva primjera, ali na većem setu različitih podataka. Također se vrši navigacija putem sakrivanja HTML elementa i radi se zapis trenutne lokacije u lokalno spremište.

Gumbi “Registracija“ i “Prijava“ rade nešto drugačije od primjera navedenih gore. Oni prvo otvaraju modal elemente. Modal elementi su zapravo skočni prozori, koji dolaze u prvi plan i zahtijevaju akciju od korisnika prije nego što je moguć nastavak korištenja aplikacije.

```
<div class="modal fade" id="loginModal" tabindex="-1" role="dialog" aria-
labelledby="loginModalLabe" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="loginModalLabe">Prijava</h5>
          <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">
            <div class="form-group">
                <label for="loginUsername">Korisničko ime:</label>
                <input type="text" class="form-control" id="loginUsername"
placeholder="Korisničko ime">
              </div>
              <div class="form-group">
                <label for="loginPassword">Lozinka</label>
                <input type="password" class="form-control"
id="loginPassword" placeholder="Lozinka">
              </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary loginModalClear"
data-dismiss="modal">Odustani</button>
```

```
          <button type="button" class="btn btn-primary loginModalClear"
id="loginSubmit" data-dismiss="modal">Prijavi se</button>
        </div>
      </div>
    </div>
  </div>
```

Ovo je primjer jednog modal prozora. Ovaj modal prozor služi za prijavu korisnika u aplikaciju. Funkcija koja se izvršava pritiskom na gumb “Prijavi se“ se također odrađuje preko javascript dijela.

```
    $('#loginSubmit').on('click', function(){

        var $username = $('#loginUsername').val();
        var $password = $('#loginPassword').val();

        $.ajax({
            type: "POST",
            url: "php/user_login.php",
            data: {username: $username, password: $password},
            success: function (data) {

                if (data!='error') {
                    sessionStorage.setItem('user',data);

                    $('#login_btn').addClass('d-none');
                    $('#register_btn').addClass('d-none');
                    $('#logout_btn').removeClass('d-none');
                    $('#uname_header').removeClass('d-none');
                    $('#uname_header').val("Dobrodošli, ".$username);

                    swal(
                        'Dobrodošli'
                    );
                } else {
                    swal(
                        'Greška',
                        'Kriva lozinka i/ili korisničko ime',
                        'error'
                    );
                }
            }
        });
    });
```

Ovaj primjer je vrlo sličan prethodno prikazanim primjerima uz iznimku što su ovdje prvo pokupljeni podaci koje je korisnik unio, te ih se preko ajax poziva poslalo na obradu u PHP skriptu. Važno je napomenuti da ako je PHP skripta uspješno odradila, u sesijsko spremište podataka se sprema identifikacijski podatak bitan za daljnju autentifikaciju korisnika.

Gumb “Registracija“ neće biti zasebno prikazan jer je način rada gotovo identičan prethodnom primjeru uz iznimku što nakon odrađivanja PHP skripte korisnik nije prijavljen

nego je registriran, tj. može se prijaviti u aplikaciju sa podacima koje je unio na ekranu registracije. Ne radi se zapis u sesijsko spremište podataka.

Gumb “Odjava“ se prikazuje samo ako je korisnik prijavljen u aplikaciju i jedina akcija mu je brisanje zapisa iz sesijskog spremišta.

```
<div style="background-color: #0e90d2; padding-left: 0; padding-right: 0;"
class='col-xs-12 navbar-fixed-top'>
            <button type="button" class="nav-link btn col-xs-3
profile_nav_layout" id="profile_tab_navigate">Moj profil</button>
            <button type="button" class="nav-link btn col-xs-3
profile_nav_layout" id="points_tab_navigate">Bodovi</button>
            <button type="button" class="nav-link btn col-xs-3
profile_nav_layout" id="moja_pitanja_tab_navigate">Moja pitanja</button>
            <button type="button" style='padding: 1.5vh 0 2vh 0;
background-color: #0e90d2; font-size: 2vh; border-radius: 0;' class="nav-
link btn col-xs-3" id="moje_ankete_tab_navigate">Moje ankete</button>
        </div>
```

Ovo je primjer HTML dijela navigacije unutar stranice profila korisnika. Neće biti prikazan javascript dio koji odrađuje akcije do kojih dolazi uslijed pritiskanja gumba navigacije. Naime akcije su načelno jednake prethodnim primjerima, jedino što je varijabilno su podaci koji se prikazuju.

```
        <div id="moja_pitanja_tab" class="d-none">

            <div style="background-color: #005E92; margin-top: 7vh;
padding-left: 0; padding-right: 0;" class='col-xs-12 navbar-fixed-top'>
                <button type="button" style='border-right: 1px solid
rgba(255,255,255,0.3)' class="nav-link btn col-xs-6 que_nav"
id="question_overview_tab">Pregled mojih pitanja</button>
                <button type="button" class="nav-link btn col-xs-6 que_nav"
id="new_question_tab">Novo pitanje</button>
            </div>

            <div id="question_overview" style="margin-top: 12vh;" class="d-
none">

            </div>

            <div id="new_question" style="margin-top: 12vh;" class="d-
none">

            </div>
        </div>
```

Ovo je primjer HTML dijela prikaza pitanja koje je korisnik kreirao. Odmah je bitno napomenuti da je HTML dio prikaza anketa identičan ovome stoga neće biti prikazan. Kao i u svim primjerima do sada, javascript dio odrađuje ajax poziv prema PHP skripti, te se na taj način vrši punjenje podacima, tj. u ovom slučaju liste pitanja kreiranih od strane korisnika.

S obzirom da nije ideja rada da se prikaže sav programski kod, nakon što su prikazane temeljne stvari i mehanike pomoću kojih zapravo cijeli sustav funkcionira, te primjeri uporabe

lokalnog i sesijskog spremišta podataka, preostalo je još istaknuti neke bitne stvari poput implementacije indeksirane baze podataka unutar aplikacije.

```
function saveProfileTempDB(id, nick, birth_year, post_number, education,
household_members, house_income, status, children, gender, work, country) {
        var indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB || window.msIndexedDB || window.shimIndexedDB;
        var open = indexedDB.open("Database", 1);

        open.onerror = function(event) {
            alert("Dogodila se greška prilikom otvaranja baze podataka!");
        };

        open.onupgradeneeded = function() {
            var db = open.result;
            var store = db.createObjectStore("ProfileObjectStore",
{keyPath: "profile_id"});
        };

        open.onsuccess = function() {
            var db = open.result;
            var tx = db.transaction("ProfileObjectStore", "readwrite");
            var store = tx.objectStore("ProfileObjectStore");

            var savingStuff = store.put({
                                        profile_id: id,
                                        nick: nick,
                                        birth_year: birth_year,
                                        post_number: post_number,
                                        country_id: country,
                                        education: education,
                                        household_members:
household_members,
                                        house_income: house_income,
                                        martial_status: status,
                                        children: children,
                                        gender: gender,
                                        employment_status: work
                                    });

            savingStuff.onsuccess = function() {
                swal(
                    'Uspjeh!',
                    'Profil spremljen!',
                    'success'
                )
                localStorage.setItem("hasUpdate", true);
            };

            savingStuff.onerror = function() {
                swal(
                    'Greška!',
                    'Profil nije spremljen!',
                    'error'
                )
            };

            tx.oncomplete = function() {
                db.close();
            };
```

```
        }
    }
```

Ovo je primjer funkcije koje se poziva u trenutku kada korisnik odluči spremiti izmjene na svom korisničkom profilu. Kao što je vidljivo te sve izmjene se zapisuju u indeksiranu bazu podataka koja će se onda naknadno sinkronizirati sa udaljenom bazom podataka, te spremiti podatke.

```
function syncTempDB(id) {
        var indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB || window.msIndexedDB || window.shimIndexedDB;
        var open = indexedDB.open("Database", 1);

        open.onerror = function(event) {
            alert("Greška prilikom otvaranja baze podataka");
        };

        open.onsuccess = function(event) {
            var db = open.result;
            var tx = db.transaction("ProfileObjectStore", "readwrite");
            var store = tx.objectStore("ProfileObjectStore");

            var getprofile = store.get(id);

            getprofile.onsuccess = function() {
                var nick = getprofile.result.nick;
                var birth_year = getprofile.result.birth_year;
                var post_number = getprofile.result.post_number;
                var education = getprofile.result.education;
                var household_members =
getprofile.result.household_members;
                var house_income = getprofile.result.house_income;
                var status = getprofile.result.status;
                var children = getprofile.result.children;
                var gender = getprofile.result.gender;
                var work = getprofile.result.work;
                var country = getprofile.result.country;

                if(status == undefined) status = '';
                if(gender == undefined) gender = '';

                $.ajax({
                    type: "POST",
                    url: "php/update_profile.php",
                    data: {
                        id: id,
                        nick: nick,
                        birth_year: birth_year,
                        post_number: post_number,
                        education: education,
                        household_members: household_members,
                        house_income: house_income,
                        status: status,
                        children: children,
                        gender: gender,
                        work: work,
                        country: country
                    },
                    success: function (data) {
                        localStorage.removeItem("hasUpdate");
```

```
                    }
                });
            };

            tx.oncomplete = function() {
                db.close();
            };
        }
    }
```

Ovo je primjer sinkronizacije indeksirane baze podataka sa udaljenom bazom podataka. Ova funkcija se odrađuje kada sustav detektira da je potrebna sinkronizacija. Detekcija se vrši automatiziranom provjerom koja se odrađuje svakih 60 sekundi, a provjerava se lokalno spremište podataka, da li u njemu postoji zapis sa ključem “hasUpdate“.

8. Zaključak

Kao što je viđeno iz ovog rada većinu spomenutih principa se isplati implementirati u moderne mrežne aplikacije. Lokalno spremište podataka, sesijsko spremište podataka i indeksirane baze podataka relativno su nova aplikacijska programska sučelja, što nažalost znači da ih ne podržavaju svi zastarjeli mrežni preglednici. Lokalno spremište podataka i sesijsko spremište podataka su blizu identični, uz razlike koje su utvrđene u prethodnim poglavljima ovog rada, te naravno pokazano na mnoštvu primjera. Indeksirane baze podataka odstupaju od tog pravila, te se bitno razlikuju od sesijskog spremišta podataka i lokalnog spremišta podataka. Sesijsko spremište podataka dostupno je samo za vrijeme trajanja sesije preglednika i briše se kada se kartica ili prozor zatvori, ali preživljava ponovno učitavanje stranice. Lokalno spremište podataka dostupno je uvijek, osim ako se naravno manualno ne obriše i preživljava sve akcije osim brisanja mrežnog preglednika sa računala ili brisanja cijelog operacijskog sustava. Indeksirane baze podataka su po svom trajanju jednake kao lokalno spremište podataka, njihova glavna značajka je baratanje objektima, te znatno veće količine spremišnog mjesta koje su na raspolaganju prilikom rada sa indeksiranim bazama podataka.

Jasno je da podaci koje mrežna aplikacija pohranjuje moraju biti stalno dostupni aplikaciji za nesmetani rad. Iako treba napomenuti kako korisnik može izbrisati te podatke, stoga treba uračunati mogućnost nestanka lokalnog spremišta podataka tokom izrade mrežne aplikacije tako da se ne oslanja u potpunosti na stalno postojanje podataka u oba slučaja.

Lokalno spremište podataka i sesijsko spremište podataka savršeni su za održavanje podataka koji nisu osjetljive prirode potrebnih unutar klijentskih skripti između stranica, primjerice: postavke, rezultati u igrama i slično. Podaci pohranjeni u lokalnom spremištu podataka i sesijskom spremištu podataka se lako mogu čitati ili mijenjati unutar klijenta, tj. klijentskog mrežnog preglednika tako da se ne bi smjeli osloniti na pohranu osjetljivih ili sigurnosnih podataka unutar aplikacija.

Indeksirane baze podataka su u biti namijenjene za čuvanje većeg broja podataka unutar klijentskog računala, te se samim time nameću za čuvanje čak i osjetljivih informacija. No to je možda razlog zašto za razliku od sesijskog i lokalnog spremišta podataka, indeksirane baze podataka ne uživaju toliku vjernost od strane mrežnih programera. Zapravo su jako rijetko korištene i nije izgledno da će se to promijeniti u nekoj skorijoj budućnosti.

Što se tiče mogućnosti, sesijsko spremište podataka i lokalno spremište podataka dopuštaju pohranu niza znamenaka, te je moguće implicitno pretvoriti primitivne vrijednosti prilikom postavljanja, naravno one će se morati pretvoriti natrag da ih koriste kao svoj tip nakon čitanja, ali ne i objekte ili nizove. Tu se javlja prednost indeksiranih baza podataka, tj. njihova

mogućnost da spremaju cijele objekte koji se onda lako mogu dohvatiti i koristiti u nastavku koda. No i dalje unatoč spomenutim benefitima mrežni programeri većinom biraju da ne koriste indeksirane baze podataka, dok su lokalno i sesijsko spremište podataka nezaobilazan alat prilikom izrade mrežnih aplikacija.

Popis literature

[1] Brown, Learning JavaScript: JavaScript Essentials for Modern Application Development, 3 edition. O'Reilly Media, 2016.

[2] D. Flanagan, JavaScript: The Definitive Guide: Activate Your Web Pages, 6 edition. O'Reilly Media, 2011.

[3] A. Shenoy, Thinking in CSS. Packt Publishing, 2014.

[4] J. Chaffer and K. Swedberg, Learning jQuery Fourth Edition, 4th Revised edition edition. Packt Publishing, 2013.

[5] T. Negrino and D. Smith, JavaScript and Ajax for the Web: Visual QuickStart Guide, 7 edition. Peachpit Press, 2008.

[6] A. Beaulieu, Learning SQL, 1 edition. O'Reilly Media, 2005.

[7] D. Bierer, PHP 7 Programming Cookbook. Packt Publishing, 2016.

[8] R. Shannon, "The History of HTML | From the HTML 1.0 spec to XHTML 1.0..." [Online]. Available: https://www.yourhtmlsource.com/starthere/historyofhtml.html. [Accessed: 10-Aug-2018].

[9] R. Clark, O. Studholme, C. Murphy, and D. Manian, Beginning HTML5 and CSS3. Berkeley, CA: Apress, 2012.

[10] "window.localStorage," MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API. [Accessed: 9-Aug-2018].

[11] J. Cash, HTTP cookie 57 Success Secrets - 57 Most Asked Questions On HTTP cookie - What You Need To Know.

[12] I. Hickson, "Web Storage (Second Edition)." [Online]. Available: https://www.w3.org/TR/webstorage/. [Accessed: 11-Aug-2018].

[13] "Introduction to events," MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events. [Accessed: 11-Aug-2018].

[14] "IndexedDB API," MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. [Accessed: 24-Aug-2018].

[15] S. Kimak and J. Ellman, "HTML5 IndexedDB Encryption: Prevention against Potential Attacks," International Journal of Intelligent Computing Research, vol. 6, no. 4, pp. 621–629, Dec. 2015.

Popis slika

Slika 1: HTML kod prvog mrežnog mjesta (*eng. website*) 6
Slika 2: Prikaz jednog kolačića u Chrome mrežnom pregledniku 19
Slika 3: Prikaz lokalnog spremišta u Chrome mrežnom pregledniku 20
Slika 4: Prikaz početne stranice 53
Slika 5: Prikaz profilne stranice 54
Slika 6: Prikaz liste korisničkih pitanja 54
Slika 7: Prikaz rezultata korisničkih pitanja 55
Slika 8: Prikaz rezultata korisničkih pitanja 55
Slika 9: Prikaz rezultata korisničkih anketa 56
Slika 10: Prikaz kreiranja pitanja 56
Slika 11: Prikaz kreiranja ankete 57
Slika 12: Prikaz liste anketa i pitanja 57
Slika 13: Prikaz odgovaranja na pitanje 58
Slika 14: Prikaz odgovaranja na anketu 58

Popis tablica

Nisu pronađeni unosi u tablici sadržaja.