ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA BIBLIOTEKE ZA GENERISANJE I MODIFIKOVANJE MREŽNE TOPOLOGIJE PRIMENOM HTML5 I JAVASCRIPT JEZIKA

– Master rad –

Kandidat:

Tijana Ilić 2011/3249

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2014.

Sažetak

U radu su ukratko predstavljene novine koje je uneo HTML5 jezik, sa posebnim osvrtom na element *canvas*. Kreirana je biblioteka za dinamičko generisanje i modifikovanje mrežne topologije primenom *canvas* elementa i dato je više praktičnih primera realizovane biblioteke. Biblioteka je napisana u *JavaScript* jeziku, a za praktične primere su upotrebljeni HTML5, CSS3 i PHP, *Ajax*, *JSON*, kao i MySQL baze podataka.

Ključne reči: HTML5, *canvas*, *JavaScript*, *Ajax*, *JSON*, PHP, MySQL, Dijkstra

SADRŽAJ

1. UVOD

Određivanje topologije telekomunikacione mreže je esencijalno za rutiranje, raspodelu saobraćaja u mreži i uopšte za monitoring mreže. Većina mreža je u konstantnom stanju promene. Tipičan primer su mobilne *ad-hoc* mreže, čija se topologija najčešće menja zbog pada linkova ili redistribucije resursa. Često je u okviru projekata i istraživanja potrebno simulirati ova dešavanja u mreži i stoga je jedan od bitnih zadataka kreiranje topologije mreže. Predmet ovog master rada je primer kreiranja mrežne topologije koristeći mogućnosti koje je uneo HTML5 jezik.

Najpopularniji brauzeri već podržavaju mnoge od novih HTML5 elemenata. Očekuje se da će novi standard, koji je evolucija HTML4 i XHTML standarda, ispraviti neke dosadašnje nedostatke i uneti niz novina i mogućnosti koje do sada nisu bile dostupne za izradu veb sajtova sem ako se nisu koristili dodaci (*plugin*-ovi) poput Flash, Java i Silverlight. HTML5 je takođe kandidat za *cross-platform* mobilne aplikacije i mnoge njegove funkcionalnosti su predviđene za korišćenje na tabletima i smart telefonima. Donosi mnoge nove funkcionalnosti, poput bolje podrške za multimedijalne sadržaje, vektorsku grafiku, *drag and drop* funkcionalnost, kao i nove elemente poput <*article*>, <*footer*>, <*header*>, <*nav*> i <*section*>, kojima se omogućava kvalitetnija izrada veb stranica. Novi element koji unosi bitnu novinu je <*canvas*>. Njegova primarna svrha je manipulacija 2D grafikom. Radom sa ovim elementom uz pomoć *JavaScript*-a moguće je kreirati dinamičku grafiku i animacije koje reaguju na korisničku interakciju, što ga čini moćnim alatom u izradi veb aplikacija.

U ovome radu formirana je biblioteka koja se može koristiti za dinamičko generisanje i modifikovanje mrežne topologije. Za prikaz mrežne topologije iskorišćen je HTML5 element <canvas>. Biblioteka je realizovana upotrebom JavaScript jezika, a za praktične primere upotrebe realizovane biblioteke iskorišćeni su HTML5, CSS3 i PHP, Ajax, kao i MySQL baza podataka. Korisnik će lako moći da je uključi u svoj sajt integracijom *JavaScript* fajlova koji će predstavljati kolekciju funkcija realizovane biblioteke.

Master rad "Implementacija biblioteke za generisanje i modifikovanje mrežne topologije primenom HTML5 i *Javascript* jezika" podeljen je u sedam poglavlja. U prvom poglavlju ukazano je na oblast, cilj, motivaciju i doprinos rada. U drugom poglavlju predstavljeni su istorijat i standardizacija HTML5 jezika, kao i novi elementi, atributi i funkcionalnosti. U trećem poglavlju je predstavljen *canvas*, HTML5 element koji je upotrebljen za prikazivanje mrežne topologije. Opisane su njegove osobine i metode koje su se koristile u radu. U četvrtom poglavlju dat je detaljan opis *JavaScript* funkcija biblioteke za dinamičko kreiranje mrežne topologije pomoću elementa *canvas*. Te funkcije pokrivaju dodavanje, ažuriranje i brisanje čvora, linka i toka korisnika, kao i grafičku predstavu čvorova i linkova. U radu su korišćeni dvosmerni linkovi. U petom poglavlju predstavljeni su primeri primene biblioteke opisane u četvrtom poglavlju, a to su ručni unos i crtanje topologije, primena Dijkstra algoritma na selektovani čvor i upisivanje i čitanje topologije iz baze podataka. Ovi primeri su generisani korišćenjem *jQuery*, *Ajax*, PHP, MySQL jezika. U šestom poglavlju, koje predstavlja zaključak rada, izložena je potencijalna primena, korist i potencijalni budući razvoj kreiranih biblioteka. U sedmom poglavlju, koje predstavlja prilog, izložen je kod pomoću kojeg su generisane slike iz poglavlja dva, koje se odnose se na *drag and*

drop funkcionalnost. Kod kompletne biblioteke i primera biće priložen uz rad u elektronskom obliku, zbog veličine.

2. HTML5

HTML (*Hypertext Markup Language*) je programski jezik koji se koristi za opis strukture veb strana. Veb strane su međusobno povezane linkovima koji su umetnuti u veb stranice. Skup ovakvih stranica naziva se hipertekst. Hipertekst se razlikuje od običnog teksta po tome što se ne čita nužno linearno (s leva na desno ili odozgo na dole), već prateći hiper veze ili linkove ka drugim veb stranicama. Jezici koji opisuju hipertekst pored HTML-a, su i SGML (*Standard Generalized Markup Language*), XML (*Extensible Markup Language*), XHTML (*Expandable HTML*) i CSS (*Cascading Style Sheets*) [1]. U ovom poglavlju biće opisan istorijat, standardizacija i novine HTML 5 verzije HTML jezika.

2.1. Istorijat i standardizacija

Za standardizaciju HTML jezika odgovoran je W3C (*World Wide Web Consortium*). HTML5 je poslednja verzija HTML jezika raspoloživa za World Wide Web. Ova nova specifikacija kao osnovu ima HTML 4.01 i XTML 1.1 specifikacije i obezbeđuje alate za kreiranje nove generacije veb sajtova.

HTML je bila zamisao Tim Berners-Lee-a, koji 1991. napisao dokument "HTML Tags", u kome je predstavio nešto manje od 200 HTML tagova, elemenata za pisanje veb strana. Jedan tag je komanda u HTML jeziku koja govori brauzeru šta i kako da uradi, odnosno na koji način da prikaže sadržaj veb stranice [1]. Tim Berners-Lee nije bio prvi koji je došao na ideju da se kao tagovi koriste reči između uglastih zagrada. Ova vrsta tagova je već postojala u SGML jeziku i umesto razvijanja potpuno novog standarda, Tim je uvideo da je korisnije da se već postojeći standard samo nadogradi. Na taj način je i HTML razvijen.

HTML 1 nikada nije postojao. Prva zvanična specifikacija je HTML-a je bila HTML 2.0, koju je objavio IETF (*Internet Engineering Task Force*). Ulogu IETF-a je potisnuo W3C, koji je objavio naredne verzije standarda. Sledeću polovinu devedesetih godina obeležio je nalet revizija specifikacije sve dok HTML 4.01 nije objavljen 1999. godine.

Posle HTML 4.01, sledeća revizija je bila nazvana XHTML 1.0. Sadržaj XHTML 1.0 specifikacije je bio identičan specifikaciji za HTML 4.01. Nisu dodati novi elementi ili atributi. Jedina razlika je bila u sintaksi. XHTML je zahtevao da se elementi i atributi pišu prema pravilima XML jezika, na kome je W3C bazirao većinu svojih tehnologija. Striktnija pravila nisu bila toliko loša jer su podstakla autore da koriste jedistveni stil pisanja. Dok su ranije tagovi mogli da se pišu samo velikim, samo malim ili kombinacijom i malih i velikih slova, XHTML 1.0 je sada zahtevao da se svi tagovi i atributi pišu samo malim slovima. Iako striknija, ova sintaksa je bila posmatrana kao najbolja praksa i bila je prihvaćena od strane veb programera.

Nakon toga, objavljen je XHTML 1.1. Dok je XHTML predstavljao HTML reformulisan kao XML, XHTML 1.1 je bio izvorni XML. To znači da nije mogao da se upotrebi mime-tip text/html u dokumentu, jer najpopularniji veb brauzer u to vreme, *Internet Explorer*, ne bi mogao da ga interpretira.

Bez obzira na sve, W3C je nastavila da razvija veb na osnovu XML-a. Počeli su da rade na XHTML 2, koji se potpuno razlikovao od XHTML 1, jer nije bio predviđen da bude kompatibilan sa postojećim sadržajima na veb-u, pa čak ni sa prethodnim verzijama HTML-a.

Kako je konzorcijum formulisao nove standarde koji su bili u suprotnosti sa potrebama veb programera, predstavnici Opere, *Apple*-a i Mozile su bili nezadovoljni. Oni su želeli da se više pažnje posveti razvoju formata koji bi dozvoljavali kreiranje veb aplikacija. Situacija je kulminirala 2004. godine, na sastanku radionica, kada je Ian Hickson, koji je radio na razvoju softvera za Operu u to vreme, predložio ideju o proširivanju HTML-a kako bi se omogućio rad na veb aplikacijama. Predlog je bio odbijen, a njegovi zagovornici su napustili W3C i formirali sopstvenu radnu grupu, WHATWG (*Web Hypertext Application Technology Working Group*).

U početku je WHATWG radio na dvema specifikacijama, *Web Forms 2.0* i Web *Apps 1.0*, koje su za cilj imale nastavak rada na HTML-u. Vremenom su se spojile u jednu specifikaciju nazvanu HTML5. Za to vreme, W3C je i dalje radio na XHTML 2, ali je taj rad tekao vrlo sporo i izgledalo je nemoguće veb preorijentisati na XML, pa je 2007. godine W3C odlučio da počne sa razvojem nove specifikacije zasnovane na radu WHATWG radne grupe, a koja će se zvati HTML 5 (sa razmakom u nazivu, koga nema u specifikaciji WHATWG radne grupe).

WHATWG je 2008. objavio *First Public Working Draft*. Već tada su delovi HTML5 bili implementirani u veb brauzere iako specifikacija nije bila dobila konačnu formu. Trenutna situacija je da W3C radi na jedinstvenom definitivnom standardu, koji je 2012. određen za kandidata za predloženi standard (eng. *Candidate Recommendation*) a 2014. treba i da postane predlog za HTML5 (engl. *Recommendation*), dok WHATWG nastavlja svoj rad na HTML kao na živom standard, standardu čiji razvoj još nije završen i koji je samim tim podložan promenama. 2011. godine WHATWG je "*HTML5: Living Standard*" preimenovala u "*HTML: Living Standard"*). Kao celina, HTML5 se takođe oslanja na ostale tehnologije, kao što je CSS3 i JavaScript [2][3].

2.2. Novi elementi i funkcionalnosti

Osnovni principi HTML5 su kompatibilnost sa prethodnim verzijama, jednostavnost, sigurnost, interoperabilnost i upravljanje greškama. Koliko je HTML5 jednostavniji govori već prva naredba koja određuje verziju jezika i njegovu varijantu i koja se naziva *DOCTYPE* (*Document Type Declaration*). U XHMTL 1.*0* ona glasi *<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">* , dok je u HTML5 znatno pojednostavljena: *<!DOCTYPE html>*. HTML5 je dizajniran da radi na različitim uređajima i platformama. Obogaćen je velikim brojem novih elemenata i funkcionalnosti. Predstavljena je grupa novih elemenata za bolje označavanje strukture veb strana, grupa semantičkih elemenata. U semantičke elemente spadaju, na primer, *header*, *nav* i *footer*, za opisivanje područja gde se pojavljuju naslov i logo, gde je navigacioni meni, i područja gde su istaknuta autorska prava, respektivno. Ovim je standardizovana učestala praksa da se navedena područja veb strana kreiraju pomoću *div* elementa i *id* atributa. Pored semantičkih elemenata, uvedeni su i novi tipovi polja za unos podataka dostupni u veb formama, kao što je polje za unos telefonskog broja ili polje za unos datuma. Predstavljeno je i nekoliko elemenata za interaktivni sadržaj kao što su video, audio i *canvas* i SVG (*Scalable Vector Graphics*). HTML5 uključuje i nekoliko aplikaciono programskih interfejsa (API), kao što su *drag and drop*, određivanje geografske pozicije i skladištenje lokalnih podataka [4].

2.2.1. Semantički elementi

Jednu od novih grupa HTML5 elemenata čine semantički elementi. Ovi elementi služe za definisanje strukture veb strane, pa se sada tako umesto <*div id="nav"*>, <*div class="header"*>, ili <*div id="footer"*>, može jednostavnije pisati <*nav*>, <*header*> i <*footer*>. Svi semantički elementi potpadaju pod <*body*>…</*body*> kontejner. Nijedan od njih nema poseban dizajn ili boju, već im je svrha da olakšaju developeru da razume kojoj logičkoj celini pripada neki HTML sadržaj. Dok se ne stilizuju pomoću CSS-a, svi elementi izgledaju jednolično i monotono. U semantičke elemente spadaju:

- <*header*>
- <*nav*>
- <*section*>
- <*article*>
- <*aside*>
- <*figure*>
- <*figcaption*>
- <*footer*>
- <*details*>
- <*summary*>
- <*mark*>
- <*time*>

Element <*section*> definiše odeljak u nekom dokumentu, uključujući i naslov odeljka. Element <*article*> je kompletan, zaseban sadržaj dokumenta ili stranice. To može biti post na nekom forumu ili blogu, novinski članak, komentar posetioca sajta ili bilo koji drugi nezavisan deo sadržaja. Ako uzmemo novine kao analogiju, one imaju više odeljaka (*sections*), kao što su sport, politika, zabava i u okviru svakog odeljka nalazi se više članaka (*articles*). Isto tako, <*section*> element se može sastojati iz više <*article*> elemenata. Element <*nav*> definiše grupu navigacionih linkova. Element <*aside*> predstavlja deo teksta koji je odvojen od glavnog sadržaja i obično se nalazi u okviru <*article*> elementa. Da bi se <*aside*> nalazio sa strane glavnog teksta, neophodno je primeniti CSS. Tag <*figure*> služi za specificiranje sadržaja koji je povezan sa glavnim tokom. Njegova pozicija je nezavisna u odnosu na glavni tok i njegovo odsustvo ne utiče na tok dokumenta. Njime se označava ilustracija, dijagram, fotografija, tabela i slično. Podelement (*child element*) elementa <*figure*> je element <*figcaption*> koji predstavlja tekst koji opisuje sadržaj <*figure*> elementa.

2.2.2. Novi atributi i elementi u okviru formulara

Jedan od elemenata za unos podataka u HTML jeziku je <*input*>. HTML5 uvodi nekoliko novih tipova koji će biti predstavljeni u ovom odeljku. Biće objašnjeni i neki od novih elemenata formulara i atributa <*form*> i <*input*> elemenata.

i) Novi tipovi input polja

HTML5 ima nekoliko novih tipova polja za unos podataka u okviru formulara, koji ograničavaju podatke koje korisnik unosi na tačno određeni format. Na ovaj način može da se obezbedi automatska validacija. Na primer, u neko polje za unos podataka (input polje) može se zahtevati unos samo brojeva, neki brojni opseg, datum, email adresa ili URL (*Uniform Resource*

Locator). Ukoliko korisnik unese podatak u formatu koji nije specificiran atributom, brauzer će to prepoznati i pojaviće se *pop-up* prozor koji će upozoriti korisnika da treba da koristi određeni format. Na žalost, ne podržavaju svi brauzeri sve tipove i u tom slučaju će se input polja ponašati kao obična tekstualna polja. Tada je neophodno obezbediti validaciju preko *JavaScript*-a ili u programskom kodu veb servera. Novi tipovi input polja su:

- *color*
- *date*
- *datetime*
- *datetime-local*
- *email*
- *month*
- *number*
- *range*
- *search*
- *tel*
- *time*
- *url*
- *week*

Atribut *type="number"* ograničava unos u input polje samo na numeričke vrednosti. Veb brauzeri koji podržavaju ovu funkcionalnost će sprečiti slanje nenumeričkih podataka ka serveru. Moguće je zadati i minimalnu i maksimalnu vrednost opsega i korak. Ako je broj van specificiranog opsega ili nije deljiv sa korakom, biće odbačen sa *pop-up* porukom. Kod za input polje tipa *type="number"* i dodatnim atributima je sledeći:

```
<input type='number' name='field'
value='number_value' min='number_value' max='number_value'
step='number_value'>
```

Atributi *max*, *min*, *step* i *value* su numeričke vrednosti. Atribut *max* je maksimalna vrednost koja sme da se unese u polje, *min* je minimalna vrednost, *step* je interval, a *value* je *default*-na vrednost. Bilo koja kombinacija atributa *min*, *max* i *step* je opciona, a *step* je jednak nuli, ukoliko vrednost *min* nije specificirana. Kada se input polje tipa *number* aktivira, tj, kada se klikne u ovo polje, sa desne strane polja pojavljuje se par strelica, jedna na gore i druga na dole, za kontrolu unosa. Ovo polje podržavaju sve novije verzije *Google Chrome* i *Opera* brauzera.

Atribut *type="date"* ograničava unos u input polje na vrednosti formatirane kao datum, npr. u formatu yyyy-mm-dd. Atributi input polja u HTML5 koji su vezani za datum su još i *month*, *week*, *time*, *datetime* i *datatime-local*. Atribut *datatime* omogućava korisniku da odabere datum i vreme (sa vremenskom zonom), *datatime-local* je za datum i vreme (bez vremenske zone), *month* je za mesec i godinu, *week* je za nedelju dana i godinu (bez vremenske zone), dok vreme omogućava izbor vremena u formatu mm:hh.

Email tip, *type="email"*, se koristi za input polja u koja treba da se unese email adresa. U slučaju da se ne unese niz karaktera koji sadrži karakter "@", brauzer će izbaciti upozorenje "Molim vas unesite email adresu" ("*Please enter an email address*"). Validaciju email adrese podržavaju sve novije verzije brauzera *Chrome*, *Firefox* i *Opera*.

Za input polja koja treba da sadrže telefonski broj može se koristiti *type="tel"*. Ova polja kao ulazne karaktere prihvataju i "+","-" i *space*. Kada veb forma zahteva broj telefona kao ulazni podatak, korisnici desktop ili laptop računara mogu jednostavno da ukucaju brojeve preko tastature, međutim u mobilnim veb brauzerima nema svrhe prikazati virtuelnu tastaturu sa svim karakterima, već je dovoljno prikazati samo numeričku tastaturu sa karakterima vezanim za telefonski broj. Ovu funkcionalnost podržavaju sve poslednje verzije *iOS Safari* i *Android* brauzera.

Input polja koja imaju atribut *type="color"* predstavljaju polja za selekciju boje. Klikom na ovo polje otvara se prozor sa spektrom boja gde korisnik može vizuelno da izabere željenu boju a ta boja se šalje ka serveru kao heksadecimalna vrednost RGB (*Red Green Blue*) modela. Ovaj atribut podržavaju brauzeri *Opera*, *Mozzila* i *Chrome*, ali je svakako dobro imati i *JavaScript* kod koji zamenjuje ovu funkcionalnost u slučaju da je neki brauzer ne podržava. Tako na primer postoji *jQuery* dodatak (*plugin*) koji se zove "*ColorPicker"* i koji se može upotrebiti u tu svrhu.

Atribut *type="range"* koristi se za input polja u koja treba da se unese vrednost iz nekog brojevnog opsega. Brauzeri koji podržavaju ovakvo input polje, prikazuju ga kao opseg sa klizačem koji može da se pomera levo i desno. Vrednost u input polju se menja u zavisnosti pozicije klizača. Kod za input polje tipa *type="range"* sa dodatnim atributima je:

```
<input type='range' name='field'
 value='number_value' min='number_value' max='number_value'
 step='number_value'>
```

Vrednosti atributa *value*, *min*, *max* i *step* su konkretne brojne vrednosti i imaju isto značenje kao kod input polja tipa *number*. Ako ovi atributi nisu zadati, podrazumeva se da je *min*=0, *max*=100 i *step*=1. Atribut *range* podržavaju sve novije verzije brauzera *Safari*, *Google Chrome* i *Opera*.

Atribut *type="url"* se koristi za ograničavanje unosa u input polje na vrednosti formatirane kao apsolutni HTTP ili HTTPS Internet adresa. Ukoliko format unosa nije dobar prikazaće se upozorenje "Molim vas unesite URL" („*Please enter a URL*") ili "Morate uneti validan URL" („*You must enter a valid URL*"). Ovaj atribut podržavaju novije verzije brauzera *Google Chrome*, *Firefox*, *Opera*, *iOS Safari* i *Internet Explorer*.

Atribut *type="search"* se koristi za polja za pretragu i ponaša se kao obično tekstualno polje sa vrlo malim kozmetičkim promenama. Naime, brauzeru *Chrome*, input polju sa desne strane pojavljuje se "x", koji kad se klikne da njega briše uneti tekst u dato polje. Sem toga, *MAC OS X Safari* prikazuje ovo polje zaobljeno, što je u skladu sa korisničkim interfejsom *Apple* proizvoda, kao što je na primer *iTunes*. Ovo polje podržavaju *Safari*, *Chrome*, *Firefox* i *Opera*.

ii) Novi elementi formulara

Novi elementi formulara su <*keygen*>, <*output*> i <*datalist*>. Element <*keygen*> predstavlja generator javnog i privatnog ključa u okviru formulara. Javni ključ se šalje sa formularom ka serveru, a privatni ključ se čuva lokalno u brauzeru. Privatni ključ se može upotrebiti za generisanje sertifikata klijenta u klijent-server komunikaciji što može da se koristi za autentifikaciju korisnika u budućnosti. Element <*output*> je mesto u formi gde će biti prikazan rezultat kalkulacije nekog skripta.

Ova dva elementa su podržana u brauzerima *Mozzila*, *Chrome*, *Safari* i *Opera*. Element <*datatalist*> je sličan tagu <*select*>, s tim što će se cela ili isfiltrirana lista pojaviti tek kada korisnik počne da kuca u input polje. Korisnik tada može da izabere vrednost koja će biti prikazana u input polju. Radi u brauzerima *Chrome*, *Firefox*, *InternetExplorer* i *Opera*. Kod prikazan ispod je primer

kako se može iskoristiti <*datalist*> element. Atribut *list* <*input*> polja se odnosi na <*datalist*> element koji sadrži predefinisane vrednosti input polja i kada korisnik klikne u input polje, pojaviće se padajuća lista sa, u primeru navedenim nazivima brauzera, između kojih korisnik može da odabere jedan:

```
<div>Izaberite brauzer iz liste brauzera:</div>
<input list="browsers" />
<datalist id="browsers">
<option value="Chrome">
<option value="Firefox">
<option value="Internet Explorer">
<option value="Opera">
<option value="Safari">
</datalist>
```

iii) Novi atributi formulara

Novi atributi za <*form*> i <*input*> element koji će biti objašnjeni u ovom odeljku su: *required*, *placeholder*, *autocomplete*, *form* i *x-webkit-speech*.

Kako bi korisnik bio u obavezi da u neko input polje unese vrednost, može se iskoristiti atribut *requred (<input requred>)*. Brauzeri koji ga podržavaju će prikazati poruku da je polje obavezno ako korisnik pokuša da pošalje podatke serveru, a da prethodno nije popunio dato polje. Ovu funkcionalnost podržavaju sve poslednje verzije brauzera *Chrome*, *Firefox* i *Opera*.

Atribut input polja *placeholder* je tekst u input polju kojom se korisniku daje savet kako da formatira ulazni podatak. Kada korisnik klikne u ovo polje, *placeholder* tekst nestaje, dozvoljavajući korisniku da kuca. Ovu funkcionalnost podržavaju sve poslednje verzije brauzera *Safari*, *Chrome*, *Firefox* i *Opera*.

Atribut *autocomplete* specificira da li će brauzer zapamtiti vrednosti iz formulara pošto su one poslate serveru. Kada se formular ponovo učita i pošto korisnik ukuca prvi karakter, za to input polje pojaviće se lista ponuđenih prethodno korišćenih vrednosti koje se poklapaju sa ukucanim karakterom. *Autocomplete* može da se odnosi na pojedinačno polje u okviru formulara, a može se odnositi i na čitav formular. *Autocomplete* isključen za ceo formular:

```
<form ... autocomplete='off'>
<input type='text' ... >
</form>
```

Ovaj atribut je uključen prema osnovnim podešavanjima, ali bi ga trebalo onemogućiti za sigurnosna tekstualna polja, kao što je na primer korisničko ime. Ako je *autocomplete* bio uključen za formular i kasnije je odlučeno da se isključi, atribut ne treba da se deaktivira samo za formular već i za svako input polje. Razlog za to je što bi *autocomplete='off'* za formular onemogućio keširanje novih vrednosti, ali bi se prethodno keširane vrednosti i dalje pojavljivale u padajućim listama kao ponuđene vrednosti. Brauzeri koji podržavaju ovaj atribut su *Internet Explorer*, *Mozzila*, *Google Chrome* i *Safari*.

Dok u ranijim verzijama to nije bilo moguće, u HTML5 jeziku input polje se sada ne mora koristiti u okviru elementa formulara. Potrebno je samo pomoću atributa input polja *form* povezati input polje sa formularom kome pripada, kao u sledećem kodu:

```
<form ... id='formid'>
</form>
<input ... name='input1' form='formid'>
```

Na ovaj način, jedno input polje može pripadati većem broju formi. Ovaj atribut funkcioniše u poslednjim verzijama brauzera *Chrome*, *Firefox* i *Opera*.

Google Chrome veb brauzer podržava govor kao ulazni podatak i njegovo prikazivanje u obliku teksta u tekstualnim poljima. Iako ovo još uvek nije deo zvanične HTML5 specifikacije, da bi se aktiviralo prepoznavanje govora <*input*> elementu se dodaje atribut *x-webkit-speech*, kojim se signalizira brauzeru da korisnik može da popuni polje pomoću govora:

```
<input type="text" x-webkit-speech>
```

Kada je govor kao ulazni podatak omogućen, element će imati sličicu mikrofona na desnoj strani input polja. Snimanje govora pokreće se klikom na ovu ikonicu. Tip atributa može biti *text*, *number*, *telephone* i *search*. Govor kao ulazni podatak za sada podržava samo *Google Chrome* počev od verzije 11.

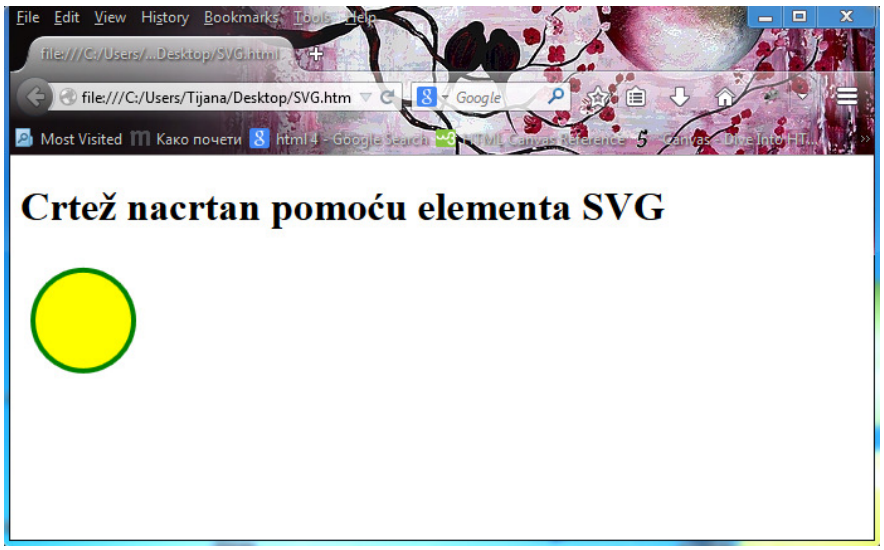
2.2.3. Canvas i SVG

HTML5 donosi i dva nova elementa za crtanje grafike, koji su zamena za *Flash plugin*. To su *Canvas* i SVG. *Canvas* je baziran na rasterskoj, a SVG na vektorskoj grafici. Rasterska grafika ili bitmap je podatak koji predstavlja pravougaonu mrežu piksela, a vektorska grafika je način prikazivanja slike pomoću geometrijskihh oblika kao što su tačke, linije, krive i poligoni, a koji su temeljeni na matematičkim jednačinama.

Element <*canvas*> omogućava dinamičko crtanje i prikazivanje grafikona, crteža i animacija. Predstavlja samo kontejner za crtanje, zbog čega je prilično zavisan od jezika za pisanje skripti, najčešće *JavaScript*-a. Ima nekoliko metoda za crtanje putanja, pravougaonika, krugova, teksta i dodavanje slika. Dodavanjem tagova <*canvas*></*canvas*> na veb stranicu, kreira se pravougaoni prostor na stranici, koji je po osnovnim podešavanjima širine 200px i visine 150px, ali se pomoću atributa širine (*width*) i atributa visine (*height*) njegove dimenzije mogu menjati. Preporučljivo je elementu dodati globalni atribut *id* kao način identifikacije *canvas* objekta u JavaScriptu. Da bi se *canvas* programerski koristio mora se prvo dohvatiti njegov contekst pomoću kog se izvode sve operacije sa njim. Tag <*canvas*> podržavaju *Internet Explorer* 9+, *Firefox*, *Opera*, *Chrome* and *Safari*. Više detalja o elementu *canvas* će biti izloženo u trećem poglavlju.

SVG, preporuka W3C, je jezik za opisivanje dvodimenzionalne vektorske grafike korišćenjem XML jezika. Svaki element SVG slike je element u DOM-u (*Document Object Model*) pa se tako može njime manipulisati kao delom DOM-a, što znači da se element može dodati, promeniti ili ukloniti direktno iz DOM-a. Svaki element i svaki atribut se može iskoristiti za animaciju. SVG elementi se mogu stilizovati pomoću CSS-a, kao bilo koji drugi HTML element. Korišćenjem sledećeg koda dobija se slika 2.2.3.1.

```
<!DOCTYPE html>
<html>
<body>
<h1>Crtež nacrtan pomoću elementa SVG</h1>
<svg width="100" height="100">
     <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
fill="yellow" />
</svg>
</body>
</html>
```



Slika 2.2.3.1. Crtanje kruga pomoću elementa SVG

Platno na kome se crta definisano je pomoću <*svg*></*svg*> tagova. Atributi *width* i *height* su širina i visina SVG platna. Element <*circle*> se koristi za crtanje kruga. Njegovi atributi *cx* i *xy* definišu koordinate x i y centra kruga. Ako su izostavljeni, znači da je centar u tački (0,0). Atribut *r* predstavlja poluprečnik kruga. Atributi *stroke* i *stroke-width* definišu boju i debljinu kružnice, a atribut *fill* se odnosi na boju kojom je krug ispunjen.

SVG je jezik za opisivanje 2D grafike pomoću XML jezika, *canvas* služi za crtanje 2D grafike pomoću *JavaScript*-a. SVG je baziran na XML-u, što znači da je svaki element SVG-a dostupan u DOM-u i se se može iskoristiti za iniciranje *JavaScript* događaja, što nije moguće implementirati za elemente *canvas*-a. U SVG-u, svaki oblik se pamti kao objekat i ako se atribut SVG objekta promeni, brauzer će automatski ponovo iscrtati taj oblik. U *canvas*-u se crta piksel po piksel i nakon što se promeni pozicija nacrtane grafike, čitav prostor *canvas*-a se mora ponovo iscrtati. Povećavanjem veličine prostora na kome se crta, vreme potrebno da se iscrta slika na *canvas*-u se povećava, zato što je potrebno iscrtati više piksela. Sa povećanjem broja objekata isctanih na platnu SVG-a, povećava se vreme potrebno da se slika prikaže jer je sve što previše koristi DOM sporo. Crtež na *canvas*-u se ne može povećati na veću rezoluciju bez gubitka kvaliteta, što nije slučaj sa SVG slikom. Bogatije manipulisanje tekstom pruža SVG nego *canvas* API. Slika u *canvas*-u se može sačuvati u .png ili .jpg formatu, dok SVG element ne pruža istu funkcionalnost. *canvas* je pogodniji za razvoj igrica, dok SVG nije. SVG je pogodan za aplikacije za iscrtavanje velike površine, npr. za *Google* Mape.

2.2.4. Elementi medije u HTML5

Novi tagovi u HTML5 su <*audio*> i <*video*>. Uz pomoć njih je moguće postaviti audio i video sadržaj na veb sajt bez potrebe za dodatkom (*plugin*). Kompletna specifikacija video elementa može se pronaći na http://www.whatwg.org/specs/web-apps/%20current-work/multipage/the-video-element.html.

Pre HTML5, audio i video sadržaji su se mogli puštati samo preko dodatka *Adobe Flash*. Suočavajući se sa problemom pouzdanosti, sigurnosti i performansi koji je pratio korišćenje *Flash cross-platform* okruženja, veb programeri su počeli da ispituju prednosti i mane korišćenja dodatka *Flash*, naročito na mobilnim uređajima. Zaključili su da bi se duže trajanje baterije, hardversko

video dekodiranje i vektorska grafika mogli ponuditi od brauzera direktno, na osnovu HTML5 standarda. Element <*video*> definiše gde će se na veb stranici pojaviti video sadržaj:

```
<video src='moviefile.format' width='320' height='240' controls autoplay>
Tekst za slučaj da video tag nije podržan
</video>
```

Atribut *src* označava lokaciju video fajla koji se pušta. Još jedan način da se definiše video sadržaj je pomoću <*source*> taga:

```
<video  width="320" height="240" controls loop muted>
<source src="moviefile.mp4" type="video/mp4">
<source src="moviefile.ogg" type="video/ogg">
<source src="moviefile.webm" type="video/webm">
Tekst za slučaj da video tag nije podržan
</video>
```

Element <*source*> se koristi za specificiranje više audio/video fajlova kao alternativu za slučaj da korišćeni brauzer ne podržava neki video/audio format. Atribut *controls* omogućava video kontrole kao što su *play, pause, volume. Width* i *height* su horizontalna i vertikalna dimenzija video plejera. Strogo je preporučljivo koristiti *width* i *height* atribute, zato što će tada prostor na veb stranici biti rezervisan za video čim se stranica učita. U suprotnom, veb brauzer neće imati informaciju o dimenzijama videa, neće rezervisati prostor za njega i kao rezultat izgled stranice će se menjati kako se video sadržaj bude učitavao. Atribut *autoplay* znači da će video početi automatski da se prikazuje čim se stranica učita, *loop* da će se po završetku ponovo početi da se prikazuje i to iznova i iznova, a *muted* da će audio biti isključen tokom prikazivanja videa.

Element <audio> definiše gde će se na veb stranici pojaviti objekat za puštanje audio sadržaja. Isto kao i za video element, postoje dva načina da se definiše lokacija audio fajla, u okviru samog elementa pomoću *src* atributa, ili pomoću *src* atributa u okviru <source> taga.

```
<audio controls autoplay loop>
<source src="audiofile.ogg" type="audio/ogg">
<source src="audiofile.mp3" type="audio/mpeg">
<source src="audiofile.wav" type="audio/wav">
Tekst za slučaj da browser ne podržava audio tag
</audio>
```

Uz <*audio*> tag mogu se koristiti atributi *controls*, *autoplay* i *loop*, koji imaju imaju isto značenje kao i za <*video*> tag.

2.2.5. Geolokacija

Geolokacija omogućava brauzeru da odredi lokaciju korisnika na osnovu nekoliko mehanizama. Ako je GPS (*Global Positioning System*) dostupan (kao što je slučaj na mnogim smart telefonima) dobiće se lokacija sa tačnošću od nekoliko metara. Ako nema GPS pristupa onda se lokacija može odrediti pomoću baznih stanica ili *WiFi hub*-ova, ali sa manjom tačnošću. U najvećem broju slučajeva, da bi se zaštitila privatnost, Geolokacijski API će zahtevati od korisnika da potvrdi zahtev za određivanjem lokacije. Da bi se odredila lokacija korisnika, poziva se metoda ***getCurrentPosition()***, koja ima dva parametra. Prvi parametar je funkcija koja se poziva ako je lokacija pronadjena, a drugi parametar je funkcija koja se poziva ako iz nekog razloga lokacija nije određena. Za slučaj da je brauzer pronašao lokaciju, metoda će vratiti geografsku širinu, geografsku

dužinu i nadmorsku visinu, kao i parametar tačnosti. Ako se dogodila greška, ispitaće se nekoliko uslova za pojavu greške.

2.2.6. Lokalno skladište

Od 1994. godine, od kada je prvi put predstavljen, HTTP kolačić (*cookie*) je dugo godina bio jedina opcija za skladištenje podataka u veb brauzeru-u. Sa razvojem HTML5, napravljen je značajan pomak u lokalnom skladištenju podataka u brauzeru. Predložena je nekoliko novih novih aplikaciono progranskih interfejsa za obezbeđivanje skladišta na klijentskoj strani. Najznačajniji su Veb skladište (*Web Storage*) i Indeksirana baza podataka (*Indexed Database*).

Veb skladište predstavlja skladištenje podataka u brauzeru kao što se to ranije radilo sa kolačićima, s tim što je Veb skladište sigurniji i brži mehanizam i podaci se ne šalju sa svakim zahtevom ka serveru, nego samo kad postoji potreba za to. Takođe, omogućava skadištenje veće količine podataka u poređenju sa kolačićima (oko 5MB), i to bez negativnog uticaja na performanse veb sajta. Podaci se skladište u vidu ime/vrednost parova, koji su uvek tipa *string*. Ime koje definiše upisani podatak, a vrednost je informacija koja se želi upamtiti. Veb skladište se sastoji od dva objekta, *localStorage* and *sessionStorage*. Razlika između njih je u tome što su podaci objekta *localStorage* postojani i nakon restarta brauzera, dok se objekat *sessionStorage* resetuje nakon restarta sesije brauzera. Veb skladište podržavaju brauzeri *Internet Explorer* 8+, *Firefox*, *Opera*, *Chrome* i *Safari*. Veb skladište je neefikasan za skadištenje velikih količina podataka, pa se za objekte sa većim brojem nivoa (*nested objects*) ili za veoma dugačke stringove, preporučuje Indeksirana baza podataka.

Indeksirana baza podataka je *JavaScript* API za skadištenje objekata i obezbeđuje veb aplikaciji da ima lokalnu bazu podataka koja je uvek dostupna. Višestruka skladišta objekata mogu postojati u istoj bazi i mogu biti indeksirani dodeljivanjem primarnog ključa koji se naziva *key path*. Ovaj API podržavaju brauzeri *Firefox* 4+ i *Chrome* 11+.

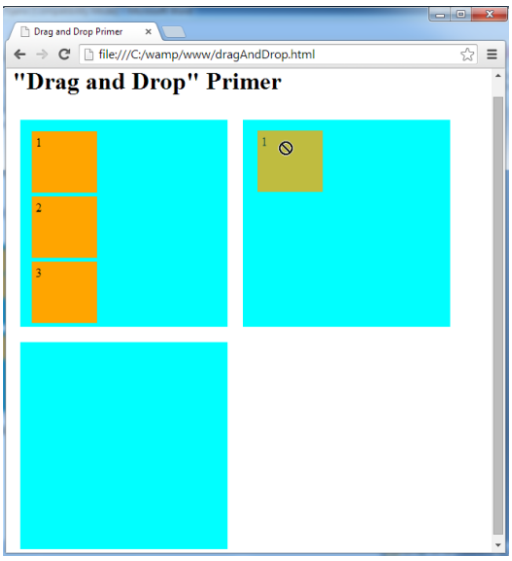
2.2.7. Drag and drop API

HTML5 standard podržava i *drag and drop* operacije, koje predstavljaju pomeranje elemenata i teksta po prozoru brauzera korišćenjem miša. Ovo je korisno za operacije korisnika kao što je na primer pomeranje artikala u korpu pri kupovini ili prilagođavanje izgleda njihove naslovne stranice pomeranjem elemenata koji se na njoj nalaze. Da bi se podržao *drag and drop* API, dodato je nekoliko atributa u HTML5, kao što je atribut *draggable*, koji se postavlja na vrednost *true* da bi se element mogao pomerati. Ipak, najveći deo *drag and drop* funkcionalnosti nekog elementa se realizuje pomoću jezika za pisanje skripti, za šta se najčešće koristi *JavaScript*. Specifikacija interfejsa za *drag and drop* se nalazi na http://dev.w3.org/html5/spec/dnd.html. Pored atributa *draggable*, sa stanovišta HTML5, *drag and drop* je podržana događajima navedenim u tabeli 2.2.7.1. Prostor u koji će se prevučeni element spustiti naziva se “meta”.

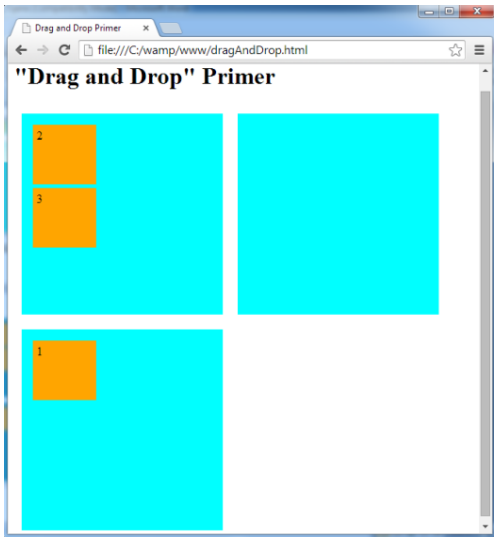
Radi boljeg objašnjenja kako funkcioniše “prevuci i pusti”, kreiran je primer, kao što je prikazano na slikama 2.2.7.1 i 2.2.7.2. Na slikama su prikazana tri <*div*> elementa koja se mogu pomerati i koja su označena brojevima 1, 2 i 3. Ovi elementi se mogu pomeriti na velike kvadrate, koji predstavljaju mete. Pokušaj pomeranja <*div*> elementa označenog brojem 1 na drugu metu rezultiraće simbolom zabrane, kao što je prikazano na slici 2.2.7.1, što implicira da meta ne prihvata element označen brojem 1. Sa druge strane, moguće je prevući 1 na treću metu, kao što je prikazano na slici 2.2.7.2. Kod pomoću kojeg su realizovani primeri sa slika 2.2.7.1 i 2.2.7.2 je naveden i objašnjen u prilogu A.

Tabela 2.2.7.1. Opis događaja operacije “prevuci i pusti”

ATRIBUT/DOGADJAJ	OPIS
dragstart	Prevuci i pusti operacija je započeta
drag	Prevuci i pusti operacija je u toku
dragenter	Element koji se pomera ulazi u prostor mete
dragover	Element koji se pomera je u prostoru mete
dragleave	Element koji se pomera napušta prostor mete
drop	Element je pomeren na prostor mete
dragend	Operacija pomeranja je završena



Slika 2.2.7.1. Zabrana *drag and drop* operacije



Slika 2.2.7.2. Uspešna *drag and drop* operacija

3. CANVAS

Canvas je inicijano kreiran 2004. godine od strane *Apple*-a, kao dodatak *Dashboard*-u i kako bi poboljšao grafiku *Safari* brauzera, a kasnije je usvojen i od strane *Firefox*-a, *Opera*-e i *Google Chrome*-a. Danas je *canvas* deo HTML5 specifikacije za sledeću generaciju veb tehnologija [5].

Većina modernih brauzera podržava *canvas* element, kao i većinu njegovih karakteristika, ali neki od njih kao što su verzije *Internet Explorer*-a pre verzije 9 ne poseduju podršku. U ovom slučaju, korisnici se porukom mogu obavestiti da njihov brauzer nema podršku i da treba da unaprede pretraživač u neku od novijih verzija. Sa druge strane, ukoliko prelazak na novu verziju nije moguć, postoji opcija korišćenja *ExplorerCanvas* skripte, koja je razvijena od strane *Google*-a i koju je samo potrebno uključiti u veb stranicu da bi *canvas* element radio i na verzijama IE pre verzije 9[6].

U ovom poglavlju će biti opisane karakteristike *canvas*-a, počev od toga kako se element uključuje u HTML stranu, pa sve do crtanja i brisanja oblika i raznih objekata u njemu.

Kao što je pomenuto u prethodnom poglavlju, *canvas* je HTML5 element koji se može ugraditi u HTML dokument sa svrhom crtanja grafike korišćenjem *JavaScript* jezika. Kao i već pomenuti audio i video elementi, *canvas* element ne zahteva nikakve dodatke (*plugin*-ove), da bi obavljao svoju funkciju [5].

Da bi se *canvas* pojavio na veb stranici potrebno je dodati sledeće HTML tagove u okviru <*body*> tagova HTML stranice:

```
<canvas id="MojCanvas" width="500" height="300"></canvas>
```

Kao što se može videti, *canvas* je definisan atributima *id*, *width* i *height*. *JavaScript* koristi atribut *id* za referenciranje elementa *canvas*, a atributi *width* i *height* su širina i visina *canvas*-a, respektivno. Ako se ne navedu širina i visina, one će biti 200 i 300 piksela, respektivno. Ovime je formiran *canvas* element koji nije uopšte vidljiv na veb strani, jer još uvek nije ništa urađeno sa 2D kontekstom samog elementa. Da bi se element učinio vidljivim, treba aplicirati stil i dodati granicu elementa pomoću atributa *border*, kao što je prikazano u kodu ispod. Na ovaj način je određen *canvas*, granice crne boje, čija je debljina jedan piksel [6]:

```
<canvas id="myCanvas" width="500" height="300"
style="border:1px solid #000000;"></canvas>
```

3.1. Crtanje pravougaonika

Na sledećem primeru biće objašnjeno crtanje jednog pravougaonika u 2D kontekstu canvas elementa.

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas" width="500" height="300" style="border:1px solid
#000000;">
<script>
```

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(20,20,150,100);
</script>
</canvas>
</body>
</html>
```

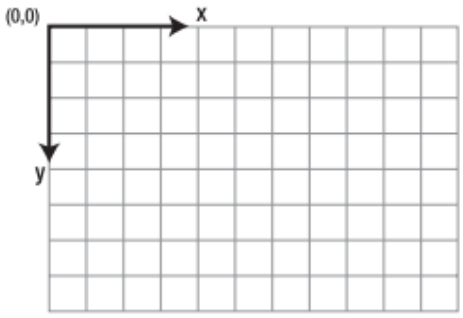
Prvo je potrebno pronaći *canvas* element na stranici preko njegovog *id* atributa koji mu je dodeljen i metode ***document.getElementById()***. U ovom slučaju identifikator je *"myCanvas"* i *canvas* se referencira na sledeći način:

```
var c = document.getElementById("myCanvas");
```

Zatim se poziva ***getContext("2d")*** metoda kojoj se kao parametar prosleđuje *"2d"* string i kojom se dobija kontekst *canvas* elementa:

```
var ctx = c.getContext("2d");
```

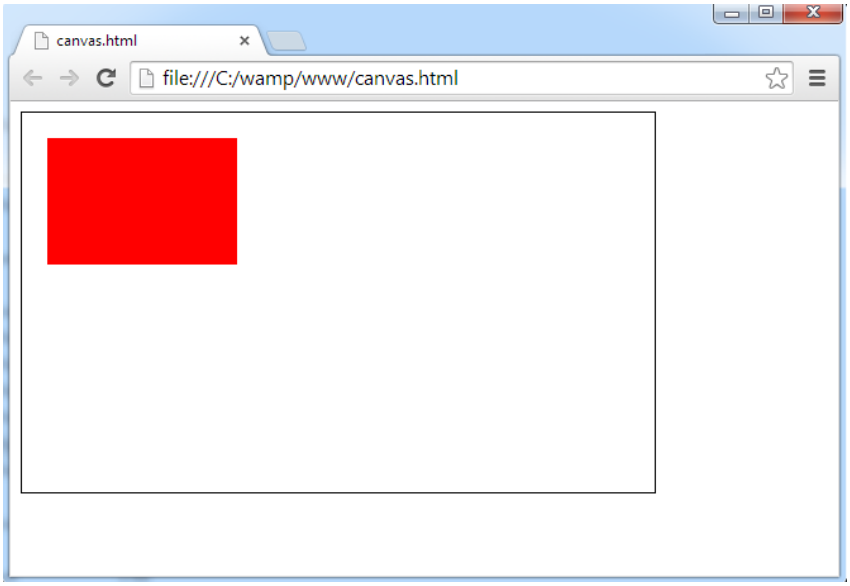
Ovaj kontekst zapravo predstavlja glavni alat za crtanje, dok *canvas* element ima ulogu omotača, koji treba da obezbedi metode i funkcionalnosti za crtanje i manipulaciju objekata. Kao i sve dvodimenzionalne platforme, ovaj kontekst koristi pravougli Dekartov koordinatni sistem sa centrom u (0,0), koji se nalazi u gornjem levom uglu, kao na slici 3.1.1. Pomeranjem udesno uvećava se vrednost x koordinate, a nadole vrednost y koordinate. Razumevanje kako je koordinatni sistem postavljen vrlo je važno za dalje crtanje svih objekata na pravilan način.



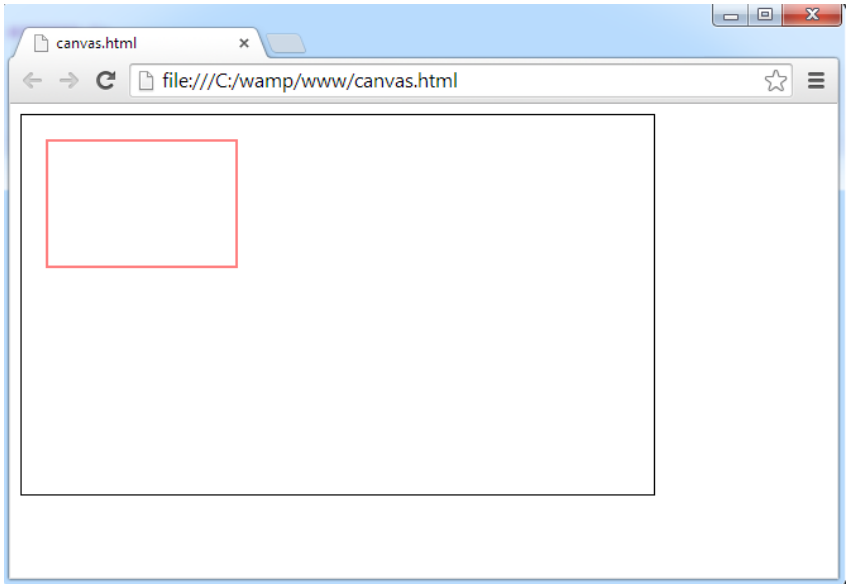
Slika 3.1.1 Koordinatni sistem elementa *canvas*

Osobina *fillStyle* može biti CSS boja, gradijent ili obrazac. Početna boja je crna, ali može biti promenjena u bilo koju boju dodavanjem stila kao rgb(crvena, zelena, plava) ili korišćenjem CSS vrednost boje kao heksadecimalnog koda (npr. #FF0000 ili reč "red"). Svaki od konteksta za crtanje će pamtiti svoja svojstva sve dok je stranica otvorena ili dok ne bude resetovan. Kako je *canvas* HTML element, moguće je koristiti CSS stilove kako bi se modifikovala njegova pozicija, dodelila slika ili boja pozadine, dodale granice i slično. Metoda ***fillRect(x, y, width, length)*** crta pravougaonik popunjen zadatim stilom i bojom. Postoje četiri argumenta potrebna za crtanje. Prva dva su (x,y) koordinate od kojih se započinje crtanje i predstavljaju poziciju gornji levi ugao pravougaonika u okviru *canvas*-a, a preostale dve su širina i visina pravougaonika. Širina se crta desno, a visina nadole od pozicije početne tačke (x,y). Osim ovoga, pravougaonik se može nacrtati i samo uokviren, bez obojene površine korišćenjem funkcije ***strokeRect(x, y, width, length)***, koja crta

pravougaonik sa okvirom, ali ne popunjava unutrašnjost. Boju, gradijent ili obrazac po kome se crta okvir određuje vrednost osobine *strokeStyle*, slično za osobina *fillStyle*. Pravougaonik uokviren crvenom bojom nacrtan u *canvas* elementu prikazan je na slici 3.1.3. Funkcija ***clearRect(x, y, width, length)*** briše piksele u specificiranom pravougaoniku.



Slika 3.1.2. Pravougaonik ispunjen crvenom bojom



Slika 3.1.3 Pravougaonik uokviren crvenom bojom

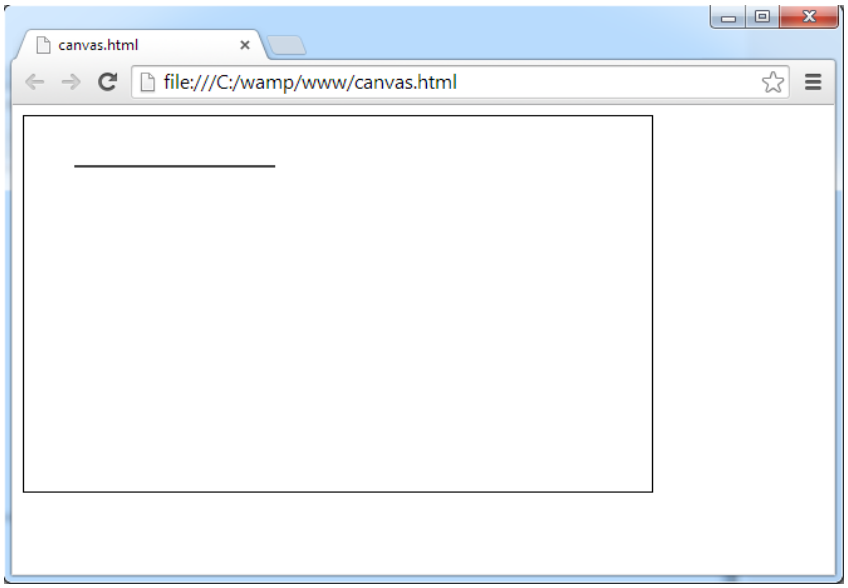
3.2. Definisanje putanja (*Path*) i crtanje linija

Kreiranje linija je malo drugačije od crtanja oblika u *canvas*-u i predstavlja se kao putanja po kojoj se crta. Definisanje ove putanje je kao skiciranje koje je podložno promenama sve do definisanja konačnog izgleda crteža i da bi se kreirala, potrebno je pozvati metodu ***beginPath()*** u 2D kontekstu.

Sledeći metod koji se poziva je ***moveTo(x, y)*** koja pomera početnu tačku linije na specificirane koordinate *(x, y)* odakle će početi crtanje, a zatim se metodom ***lineTo(x, y)*** crta linija od prethodno definisane početne tačke do koordinata zadate krajnje tačke. Dalje, metodom ***closePath()*** završava se crtanje linije. Ove metode "skiciranja" mogu se često pozivati, ali uopšte neće biti vidljive u *canvas* elementu dok se ne pozove metoda ***stroke()*** koja crta liniju prema prethodno zadatim podacima. Sve prethodno navedeno može se prikazati sledećim kodom:

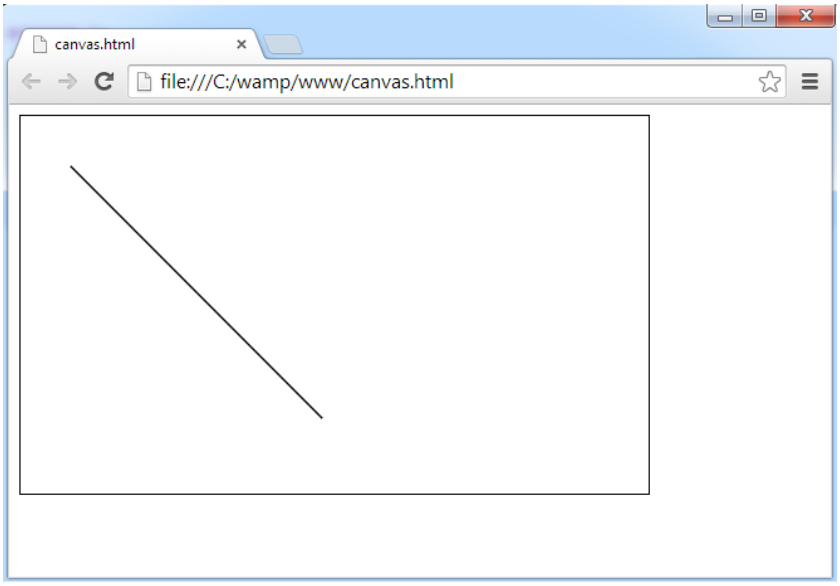
```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas" width="500" height="300" style="border:1px solid
#000000;">
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.moveTo(40,40);
ctx.lineTo(200,40);
ctx.closePath();
ctx.stroke();
</script>
</canvas>
</body>
</html>
```

Navedeni kod kao rezultat daje horizontalnu liniju, predstavljenu na slici 3.2.1:



Slika 3.2.1. Horizontalna linija

Promenom koordinata metode ***lineTo(40,40)*** na ***lineTo(240,240)*** dobija se kosa linija kao na slici 3.2.2:



Slika 3.2.2 Dijagonalna linija

3.3. Crtanje kruga

Crtanje krugova u *canvas*-u je značajno drugačije od crtanja pravougaonika jer je krug prilično složen oblik i u *canvas*-u ne postoji poseban metod za njegovo kreiranje. Ono što postoji je metoda za crtanje luka, što krug i jeste - luk spojen na oba kraja. Sledeći kod ilustruje crtanje jednog kruga tj kružnice:

```
context.beginPath();
context.arc(230, 90, 50, 0, Math.PI*2, false);
context.fill();
```

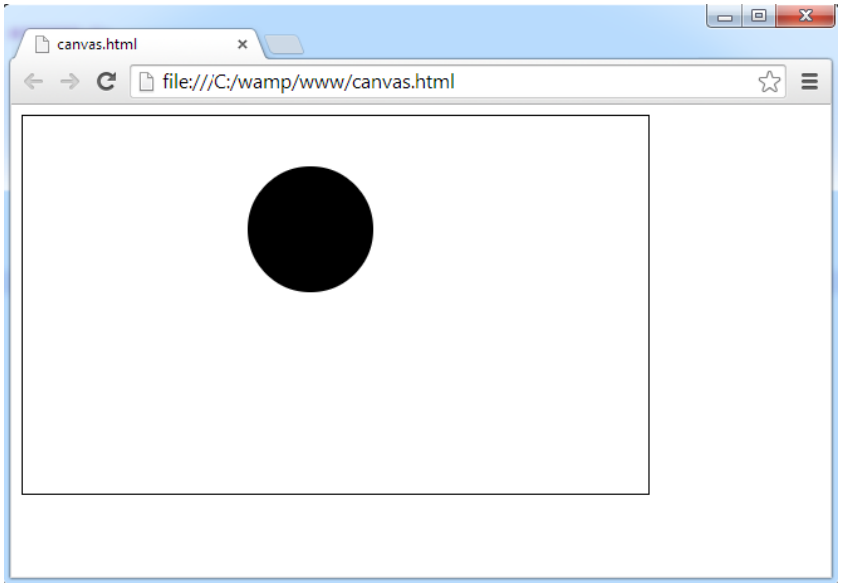
Prva linija koda koristi se za početak od koga će se crtanje luka tj. kruga započeti, dok treća linija iscrtava zadatu putanju. U drugoj liniji koda, kroz šest argumenata, definiše se sve neophodno za crtanje kruga. Prva dva su koordinate centra kruga (x,y), zatim poluprečnik ili radijus, početni ugao, završni ugao i kao poslednji argument, (*boolean*) vrednost kojom se definiše crtanje luka, a koja može biti "*true*" – u smeru suprotnom od kazaljke, ili "*false*" – u smeru kazaljke na satu. Bitno je napomenuti da su uglovi u canvasu u radijanima, a ne u stepenima. Prema tome, crtanje kruga korišćenjem funkcije za crtanje luka može se napisati čitljivije kao:

```
context.arc(x, y, radius, početniUgao, završniUgao, suprotnoOdKazaljke);
```

Ako se u brauzeru pokrene sledeći kod dobija se krug prikazan na slici 3.3.1:

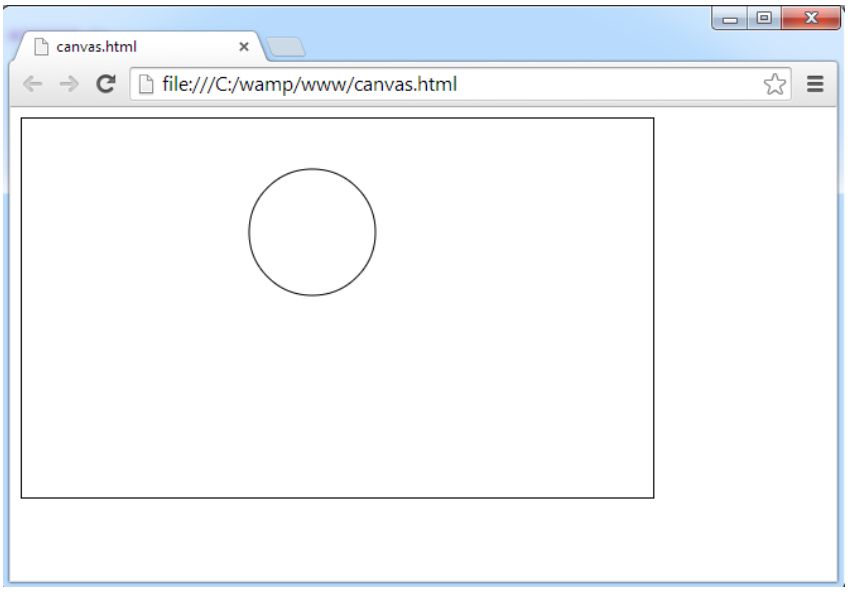
```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas" width="500" height="300" style="border:1px solid
#000000;">
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath(); // Započni putanju
ctx.arc(230, 90, 50, 0, Math.PI*2, false);  // Nacrtaj krug
```

```
ctx.fill(); // Zatvori putanju
</script>
</canvas>
</body>
</html>
```



Slika 3.3.1 Crtanje kruga

Ukoliko se umesto poziva metode ***context.fill()*** pozove ***context.stroke()*** kao rezultat dobija se kružnica prikazana na slici 3.3.2:



Slika 3.3.2 Crtanje kružnice

3.4. Crtanje teksta

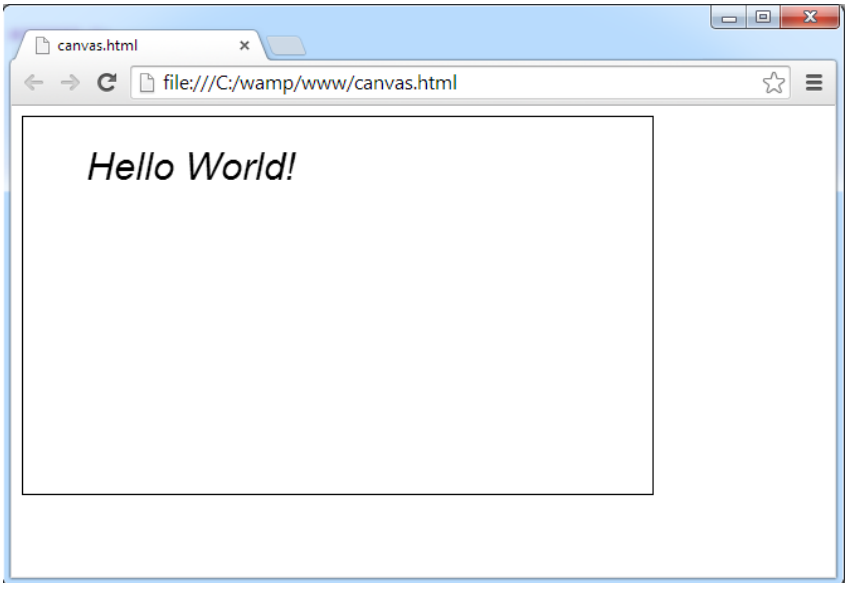
Osim mogućnosti crtanja grafike i slika, u *canvas*-u se može crtati i tekst. Međutim ovako prikazan tekst zapravo nije tekst u pravom smislu, već predstavlja sliku, što znači da ga nije moguće označiti korišćenjem miša ili tastature kao normalan tekst u HTML-u. Sa druge strane, pogodnosti ovakvog pristupa je mogućnost upotrebe različitih funkcionalnosti koje *canvas* poseduje da bi se tako iscrtan tekst transformisao. Ipak, preporuka je da se tekst, ukoliko je to moguće, uvek predstavlja HTML elementima predviđenim za ispis teksta, a zatim da se upotrebom CSS-a i *layer*-a postave preko *canvas*-a sa CSS pozicioniranjem. Poenta je u tome da je HTML i kreiran da radi sa tekstom, a *canvas* sa grafičkim elementima i slikama.

Za primer crtanja teksta će se koristiti sledeći kod, koji pokrenut u brauzeru izgleda kao na slici 3.4.1:

```
var text = "Hello World!";
ctx.font = "italic 30px Arial";
ctx.fillText(text, 50, 50);
```

Osobinom *font* se zadaju svojstva fonta kao *string*, na isti način kao i kod CSS svojstava fonta. U primeru je zadata veličina teksta u pikselima i ime font-a, kao i italic stil.

Metoda ***fillText()*** kao prvi argument ima tekst koji se iscrtava, a drugi i treći argument su koordinate u *canvas*-u gde će tekst biti iscrtan. Sličan slučaj je i sa metodom ***strokeText()*** koja ima iste parametre kao i metoda ***fillText()*** [6].



Slika 3.4.1 Crtanje teksta

4. FUNKCIJE ZA DINAMIČKO GENERISANJE MREŽNE TOPOLOGIJE

Telekomunikaciona mreža je sistem povezanih računara koji mogu međusobno razmenjivati podatke i resurse. Veza između dva računara ili računarska uređaja u mreži naziva se link. Tipično su to upredene parice (*twisted pair*), koaksijalni kablovi, optički kablovi i radio linkovi. Način na koji su uređaji povezani u mrežu predstavlja topologiju mreže. To je tačnije šematski prikaz geografskog položaja linkova i čvorova koji čine mrežu. Čvorovi mreže su hostovi (radne stanice) i mrežni uređaji. Postoji veliki broj telekomunikacionih mreža i većina tih mreža je u konstantnom stanju promene, čvorovi se pridružuju ili uklanjaju iz mreže, a linkovi se pojavljuju i nestaju. Često je u okviru projekata i istraživanja potrebno simulirati ova dešavanja u mreži i stoga je jedan od bitnih zadataka kreiranje topologije mreže.

U ovom poglavlju biće opisana biblioteka funkcija koje su napisane u *JavaScript* jeziku, a koje služe za dinamičko generisanje mrežne topologije pomoću HTML5 elementa *canvas*. Funkcije koje su bibliotekom pokrivene su dodavanje, crtanje i brisanje čvora, dodavanje, crtanje i brisanje linka i dodavanje i brisanje toka korisnika. Moguće je i ažuriranje parametara čvora, linka i toka. Parametri čvora u ovome radu su koordinate čvora i kašnjenje čvora, parametri linkova su kapacitet, kašnjenje, cena, identifikacija početnog i završnog čvora, a parametri tokova su identifikacija početnog i završnog čvora i zahtevani kapacitet. Listu navedenih parametara za čvor, link i tok je moguće lako proširiti.

4.1. Dodavanje čvora, linka i toka

Opis kreiranih funkcija započinje sa funkcijama koje opisuju tri osnovna objekta koja se koriste kao elementi mrežne topologije. Prva funkcija opisuje objekat koji se naziva ***Cvor*** i za ulazne parametre ima parametre čvora. Parametri koji opisuju čvor opisuju njegov položaj u koordinatnom sistemu *canvas*-a i kašnjenje čvora.

```
function Cvor(KoordinataX, KoordinataY, KasnjenjeCvora) {
  this.KoordinataX = parseFloat(KoordinataX);
  this.KoordinataY = parseFloat(KoordinataY);
  this.KasnjenjeCvora = parseInt(KasnjenjeCvora);
}
```

Promenljiva *KoordinataX* je X koordinata, a promenljiva *KoordinataY* je Y koordinata u 2D koordinatnom sistemu elementa *canvas*. Promenljiva *KasnjenjeCvora* označava kašnjenje datog čvora. Korisnik preko korisničkog interfejsa zadaje vrednosti ovih parametara i one se prosleđuju funkciji ***Cvor***, u kojoj se vrši konverzija argumenata koji su tipa string u format broja pomoću funkcija ***parseFloat()*** i ***parseInt()***, respektivno. Funkcija ***parseFloat*** vraća vrednost izraženu kao broj u pokretnom zarezu, koji je tipa *float*, a funkcija ***parseInt*** vraća vrednost izraženu kao celobrojan broj, koji je tipa *integer*. Konvertovani parametri *KoordinataX*, *KoordinataY* i *KasnjenjeCvora* se zatim postavljaju kao osobine objekta *Cvor*, a to su *KoordinataX*, *KoordinataY* i *KasnjenjeCvora*.

Da bi se kreirao novi objekat tipa ***Cvor,*** može se koristiti naredba:

```
var Cvor1=new Cvor (100, 200, 3);
```

U ovom primeru, promenljiva *Cvor1* je novokreirani objekat tipa ***Cvor***, koji će za koordinatu X imati vrednost 100, za koordinatu Y vrednost 200, a za kašnjenje vrednost 3. Za čuvanje niza ovakvih objekata predviđena je promenljiva *Cvorovi*:

```
var Cvorovi=[];
```

Kreiranje niza *Cvorovi* koji sadrži, na primer, tri proizvoljna elementa *Cvor1*, *Cvor2* i *Cvor3*, gde su čvorovi *Cvor2* i *Cvor3* kreirani na isti način kao *Cvor1* pomoću funkcije ***Cvor***, obavlja se pomoću metode ***push()*** i to na sledeći način:

```
Cvorovi.push(Cvor1);
Cvorovi.push(Cvor2);
Cvorovi.push(Cvor3);
```

Metoda ***push()*** dodaje elemente na kraj niza, i ako je niz *Cvorovi* prethodno bio prazan, to znači da će *Cvor1* biti element niza *Cvorovi* sa indeksom 0, *Cvor2* element sa indeksom 1 i *Cvor3* element sa indeksom 2.

Pored čvora, jos jedan element, koji je neophodan za prikazivanje mrežne topologije, je link. Parametri koji opisuju link su identifikacija početnog i završnog čvora, cena, kapacitet i kašnjenje linka, stoga će se objekti, kao i za čvor, koristiti za opis ovog elementa. Funkcija koja opisuje ovaj objekat koji se naziva ***Link*** je:

```
function Link(PocetniCvor, KrajnjiCvor,Cena,Kapacitet,KasnjenjeLinka) {
  this.PocetniCvor = parseInt(PocetniCvor);
  this.KrajnjiCvor = parseInt(KrajnjiCvor);
  this.Cena = parseInt(Cena);
  this.Kapacitet = parseFloat(Kapacitet);
  this.KasnjenjeLinka = parseInt(KasnjenjeLinka);
}
```

Ulazni parametri funkcije su parametri koji opisuju link. *PocetniCvor* i *KrajnjiCvor* predstavljaju redne brojeve čvorova koje uneti link povezuje. U radu su korišćeni dvosmerni linkovi tako da nije bitan redosled unošenja početnog i krajnjeg čvora. Svi parametri, sem parametra *Kapacitet*, su određeni celim pozitivnim brojevima, pa se parametar *Kapacitet* konvertuje iz tipa *string* u tip *float*, a *PocetniCvor*, *KrajnjiCvor*, *Cena* i *KasnjenjeLinka* se pomoću funkcije ***parseInt()*** konvertuju iz tipa string u tip integer. Posle konverzije, vrednosti ulaznih parametara se postavljaju kao vrednosti osobina objekta ***Link***.

Novi objekat tipa Link se može kreirati naredbom:

```
var Link1=new Link (1, 2, 1, 100, 2);
```

Link1 je naziv objekta tipa ***Link*** koji spaja čvorove sa identifikacijma 1 i 2, cena linka je 1, kapacitet 100 i kašnjenje 2. U primeru je pretpostavljeno da se kapacitet unosi u Mb/s. Za čuvanje niza objekata tipa ***Link*** koristi se niz objekata koji se naziva *Linkovi* i koji je deklarisan naredbom:

```
var Linkovi=[];
```

Dodavanje elemenata u niz *Linkovi* vrši se na isti način kao za niz *Cvorovi*. Niz *Linkovi* koji se sastoji od tri objekta, *Link1*, *Link2* i *Link3*, gde se *Link1* i *Link2* kreiraju na isti način kao *Link1*, dobija se naredbama:

```
Linkovi.push(Link1);
```

```
Linkovi.push(Link2);
Linkovi.push(Link3);
```

Pored čvora i linka, za opis mrežne topologije koriste se i tokovi korisnika. Oni, kao i linkovi, mogu biti jednosmerni i dvosmerni, a u ovome radu je pretpostavljeno da su dvosmerni. Funkcija koja opisuje objekat koji se naziva ***Tok*** je:

```
function Tok(PocetniCvorToka, ZavrsniCvorToka, KapacitetToka) {
  this.PocetniCvorToka = parseInt(PocetniCvorToka);
  this.ZavrsniCvorToka = parseInt(ZavrsniCvorToka);
  this.KapacitetToka = parseFloat(KapacitetToka);
}
```

Parametri koji definišu tok podataka su identifikacije dva čvora između kojih se obavlja komunikacija i željeni kapacitet saobraćaja korisničkih parova. Ulazni parametri funkcije ***Tok*** su prema tome *PocetniCvorToka* i *ZavrsniCvorToka*, kao identifikacije krajnjih čvorova, i parametar *KapacitetToka*, koji označava zahtevani kapacitet. Vrednosti parametara *PocetniCvorToka* i *ZavrsniCvorToka* se konvertuju iz tipa string u celobrojni tip podataka, a *KapacitetToka* iz tipa *string* u tip *float*, jer je predviđeno da se za kapacitet može uneti i decimalna vrednost. Posle konverzije, vrednosti parametara se postavljaju za osobine objekta tipa ***Tok***.

Novi objekat tipa ***Tok*** se može zadati sledećom naredbom:

```
var Tok1=new Tok (1, 5, 40);
```

U datom primeru *Tok1* je novokreirani objekat tipa ***Tok*** koji označava da se u mreži obavlja razmena informacija između čvorova sa identifikacijama 1 i 5 a da je kapacitet koji zahtevaju korisnički parovi 40Mb/s. Za skadištenje niza objekata tipa ***Tok*** deklarisana je sledeća funkcija:

```
var Tokovi=[];
```

Niz Tokovi se popunjava elementima na isti način kao nizovi *Cvorovi* i *Linkovi*, pomoću metode ***push()***.

4.2. Crtanje čvora, linka i toka

Za grafičku predstavu topologije mreže u ovom radu je upotrebljen HTML5 element <*canvas*>. Kao što je opisano u trećem poglavlju, *canvas* je prostor u obliku pravougaonika na HTML stranici, koji se koristi za kreiranje dinamičkih slika, u ovom slučaju za prikazivanje topologije mreže. Kanvas je dvodimenzionalna koordinatna mreža, čiji je koordinatni početak u gornjem levom uglu kanvasa, pri čemu se X koordinata povećava sa leva na desno, sa širinom kanvasa, a Y koordinata se povećava ka donjem delu, prateći visinu kanvasa. Elementu <*canvas*> uvek treba pridružiti *id* atribut, kako bi se mogao pozvati u kodu, kao i atribute *width* i *height*, koji definišu širinu i visinu kanvasa, respektivno. Da bi se pronašao <*canvas*> element, poziva se *JavaScript* metoda ***getElementById()***, a zatim se poziva njegov ***getContext("2d")*** metod.

Element kojim se započinje crtanje topologije mreže je čvor, čije se koordinate unose preko korisničkog interfejsa za dodavanje novog čvora. Da bi se novi čvor ili vise njih nacrtalo, koristi se funkcija ***CrtajCvorove()*** koja je deo biblioteke funkcija napisane u ovom radu, i kojoj je potrebno proslediti referencu na *canvas* element, odnosno *document.getElementById("MojCanvas")*, njegov kontekst, kao i niz čvorova koji treba da se iscrtaju u elementu *canvas*, kao deo mrežne topologije.

```
function CrtajCvorove(c,ctx,Cvorovi) {
```

```
  ctx.clearRect(0, 0, c.width, c.height);
  var n=Cvorovi.length
  for (i=0;i<n;i++){
    var Cvori=Cvorovi[i];
    ctx.beginPath();
    ctx.arc(Cvori.KoordinataX,Cvori.KoordinataY,20,0,2*Math.PI);
    ctx.stroke();
    ctx.textAlign="center";
    ctx.textBaseline="middle";
    ctx.strokeText(i+1,Cvori.KoordinataX,Cvori.KoordinataY);
  }
}
```

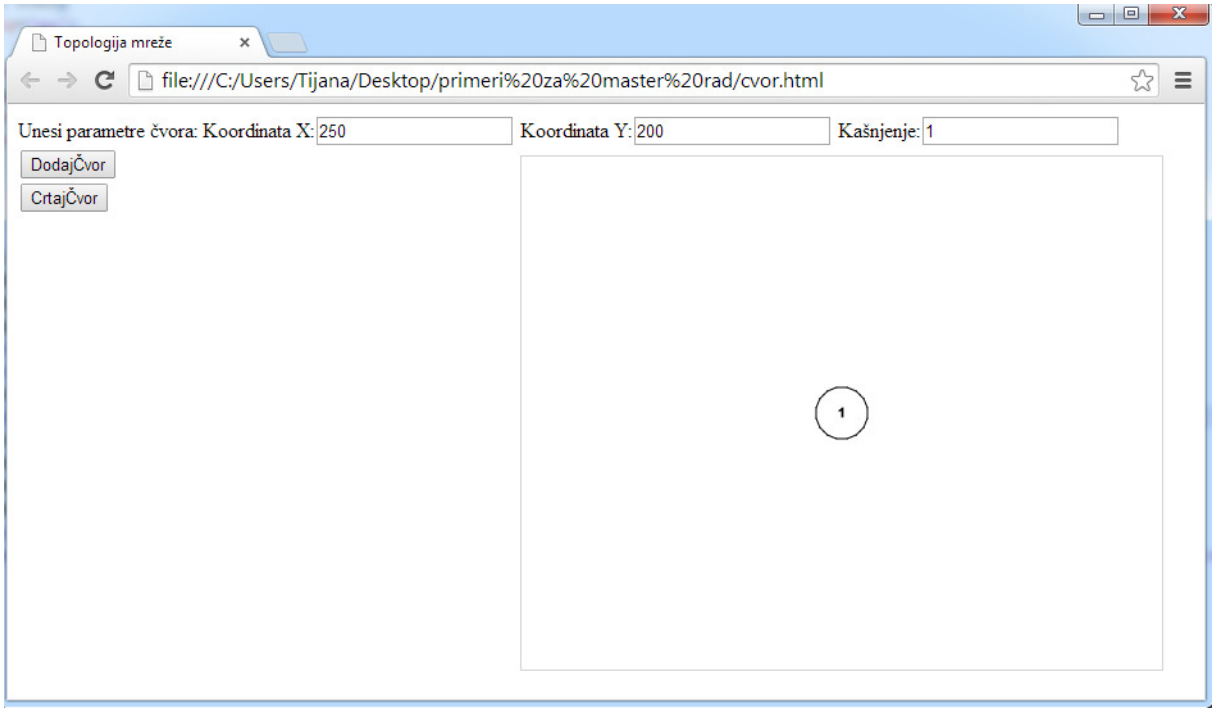
Referenca na *canvas* je određena promenljivom *c*, *ctx* je kontekst, a *Cvorovi* niz objekata *Cvorovi*. U okviru funkcije, a pre samog iscrtavanja, potrebno je obrisati sadrzaj *canvas*-a, što se radi naredbom `ctx.clearRect(0, 0, c.width, c.height); ,` a zatim odrediti dužinu niza objekata *Cvorovi*. Promenljiva *n* predstavlja dužinu niza *Cvorovi*, a određena je *JavaScript* metodom za određivanje dužine niza ***length()***. Nakon toga, funkcija prolazi kroz sve elemente ovog niza pomoću FOR petlje. U svakom prolasku kroz FOR petlju, definiše se promenljiva *Cvori*, koja predstavlja trenutni element niza *Cvorovi*, i za svaki taj element, odnosno čvor, se postavlja nova putanja metodom ***beginPath()*** i crta se krug, metodom ***arc()***, koji će predstavljati čvor u okviru mrežne topologije. Funkciji ***arc()*** je potrebno pet argumenata za konstruisanje luka. To su koordinate x i y, koje predstavljaju centar luka, zatim poluprečnik, početni ugao od koga počinje crtanje i krajnji ugao gde se crtanje završava. Kako je krug luk spojen na oba kraja, kao početni ugao prosleđuje se 0, a kao krajnji 2π radijana i na taj se dobija predstava čvora u vidu kruga. Poluprečnik kruga se zadaje u pikselima i, po ovoj implementaciji, korisnik ga ne može promeniti preko korisničkog interfejsa. Naredbe za crtanje čvora u funkciji ***CrtajCvorove()*** su:

```
ctx.beginPath();
ctx.arc(Cvori.KoordinataX,Cvori.KoordinataY,20,0,2*Math.PI);
ctx.stroke();
```

Preostalo je još da se u samom čvoru napiše i redni broj. To se postiže korišćenjem *canvas* funkcije ***strokeText()***. Za postavljanje pozicije teksta koriste se funkcije ***textAlign()*** i ***textBaseline()***. Tekst je pozicioniran da bude u centru kruga. Sama funkcija ***strokeText()*** crta tekst rednog broja čvora koji dobija kao prvi argument, na koordinatama koje su zadate kao sledeća dva argumenta. Opisane naredbe za označavanje čvora su:

```
ctx.textAlign="center";
ctx.textBaseline="middle";
ctx.strokeText(i+1,Cvori.KoordinataX,Cvori.KoordinataY);
```

Sledeći primer, na slici 4.2.1, ilustruje korišćenje prethodno opisanih funkcija za dodavanje i crtanje čvora. Za potrebe primere kreirana je posebna HTML stranica cvor.html.



Slika 4.2.1. Dodavanje i crtanje čvora u brauzeru *Google Chrome*

Na slici se može videti da korisnik u polja za unos podataka unosi parametre čvora, dakle koordinatu X, koordinatu Y i kašnjenje. Uzeto je da je *canvas* širine 500px i visine 400px. Klikom na dugme DodajČvor uneti podaci se prosleđuju u niz ***Cvorov***i, a klikom na CrtajČvor na *canvas*-u se iscrtava krug sa oznakom rednog broja unosa čvora. HTML kod koji je korišćen za generisanje prikazane stranice je:

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>Topologija mreže</title>
<script src="crtajcvor.js"></script>
<script src="lib.js"></script>
</style>

</head>
<body>
Unesi parametre čvora:
Koordinata X:<input type="number" id="X">
Koordinata Y:<input type="number" id="Y">
Kašnjenje:<input type="number" id="D">
<button type="button" onclick="DodajCvor();" >DodajČvor</button><br/>
<button type="button" onclick="CrtajCvor();">CrtajČvor</button><br/>
<canvas id="myCanvas" width="500" height="400" style="border:1px solid #d3d3d3;
position:absolute; left:400px; top: 40px; right:30px;">
</body>
</html>
```

Stranica poziva *JavaScript* fajl crtajcvor.js, u kome se nalaze funkcije koje se pozivaju klikom na DodajCvor i CrtajCvor, a to su ***DodajCvor()*** i ***CrtajCvor()***. Ove funkcije su date sledećim kodom:

```
function DodajCvor () {
var x=document.getElementById("X").value;
var y=document.getElementById("Y").value;
var d=document.getElementById("D").value;
var NoviCvor= new Cvor(x,y,d);
Cvorovi.push(NoviCvor);
}

function CrtajCvor() {
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
CrtajCvorove(c,ctx,Cvorovi);
}
```

Funkcija ***DodajCvor()*** uzima vrednosti parametara čvora i pomoću njih kreira novi objekat tipa čvor koji se naziva *NoviCvor*, a zatim ga dodaje u niz *Cvorovi*. Funkcija ***CrtajCvor()*** poziva funkciju ***CrtajCvorove()***, koja se, kao i funkcija ***Cvor()*** i niz *Cvorovi*, nalazi u biblioteci lib.js.

Pored grafičkog prikaza čvorova potrebno je prikazati i linkove da bi topologija mreže bila potpuna. Da bi se nacrtao link, odnosno linija u *canvas*-u, potrebno je zadati X i Y koordinate početne i krajnje tačke koje ta linija treba da spaja. Funkcija koja crta jedan ili više linkova, koje je korisnik zadao putem korisničkog interfejsa, je ***CrtajLinkove()***. Ulazni parametri ove funkcije su referenca na element kanvas, koja je označena promenljvom *c*, kontekst kanvas elementa, koji je označen promenljivom *ctx* i niz zadatih linkova koji je u funkciji označen sa *Linkovi*. Kod funkcije ***CrtajLinkove()*** je sledeći:

```
function CrtajLinkove(c,ctx,Linkovi)  {
  var n=Linkovi.length
  for (i=0;i<n;i++) {
    var Linki=Linkovi[i];
    var from=Linki.PocetniCvor;
    from=from-1;
    var to=Linki.KrajnjiCvor;
    to=to-1;
    var Poc=Cvorovi[from];
    var Kraj=Cvorovi[to];
    var PocX=Poc.KoordinataX;
    var PocY=Poc.KoordinataY;
    var KrajX=Kraj.KoordinataX;
    var KrajY=Kraj.KoordinataY;
    ctx.moveTo(PocX,PocY);
    ctx.lineTo(KrajX,KrajY);
    ctx.stroke();
  }
}
```

U funkciji se prvo definiše promenljiva *n*, koja predstavlja dužinu niza *Linkovi*, i čija je svrha u inicijalizaciji FOR petlje koja sledi. Funkcija svake iteracije FOR petlje je da iscrtava po jedan link. U svakom prolazu se definiše lokalna promenljiva u ciklusu, *Linki*, kojoj se dodeljuje vrednost objekta niza *Linkovi* sa indeksom *i*. Promenljive *from* i *to* su početni i krajnji čvor tekućeg linka, respektivno, i celobrojnog su tipa. Kako redni brojevi čvorova koje korisnik unosi počinju od jedan, a indeksiranje počinje od nule, vrednosti promenljivih *from* i *to* se umanjuju za jedan, da bi se pomoću njih dobile vrednosti indeksa čvorova koji se nalaze u nizu *Cvorovi* i koji sadrže podatke o koordinatama čvorova, koje su neophodne za iscrtavanje svakog linka. Čvorovi sa indeksima *from* i *to* iz niza *Cvorovi*, su dakle čvorovi između kojih treba da se nacrta link i sledeće što se radi je

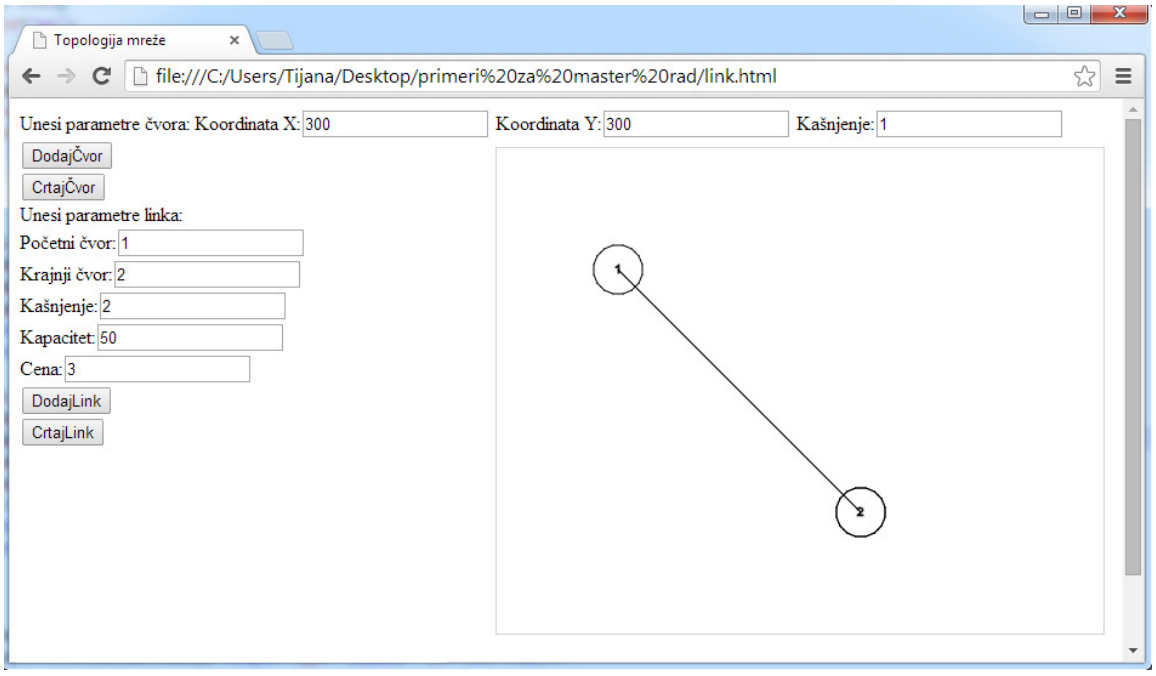
definisanje novih promenljivih, sa nazivima *Poc* i *Kraj*, kojima se dodeljuju vrednosti ta dva elementa niza objekata *Cvorovi*. *Poc* je početni čvor linka, a *Kraj* krajnji čvor linka. Promenljiva *PocX* se definiše kao koordinata X početnog čvora linka, a *PocY* je koordinata X početnog čvora linka. Ekvivalento tome, *KrajX* je predstavlja X koordinatu krajnjeg čvora linka, a *KrajY* Y koordinatu krajnjeg čvora linka. Pošto su koordinate čvorova dobijene, ostalo je samo da se link nacrta, što se izvršava sledećim trima naredbama:

```
ctx.moveTo(PocX,PocY);
ctx.lineTo(KrajX,KrajY);
ctx.stroke();
```

moveTo() metodom se definiše početna tačka linka, a ***lineTo()*** metodom krajnja tačka linka. Preostaje samo da se povuče linija između dve tačke, što se izvršava metodom ***stroke()***. Time je završeno crtanje jednog linka i prolazak kroz petlju FOR.

Slika 4.2.2 prikazuje dodavanje i crtanje linka u elementu *canvas*. Da bi se dodao link, prethodno je potrebno dodati čvorove. U primeru su dodati i nacrtani čvorovi sa koordinatama (100,100) i (300,300) i kašnjenjem 1. Da bi se kreirao link, treba popuniti polja za parametre linka i kliknuti na dugme DodajLink. Da bi se link nacrtao u elementu kanvas, treba kliknuti na dugme CrtajLink. HTML kod korišćen za generisanje slike je dodatak na HTML kod za generisanje slike 4.2.1.:

```
Unesi parametre linka:<br/>
Početni čvor:<input type="number" id="C1"><br/>
Krajnji čvor:<input type="number" id="C2"><br/>
Kašnjenje:<input type="number" id="Kasnjenje"><br/>
Kapacitet:<input type="number" id="Kapacitet"><br/>
Cena:<input type="number" id="Cena"><br/>
<button type="button" onclick="DodajLink();">DodajLink</button><br/>
<button type="button" onclick="CrtajLink();">CrtajLink</button><br/>
```



Slika 4.2.2. Dodavanje i crtanje linka u brauzeru *Google Chrome*

HTML stranica posebno kreirana za ovaj primer je nazvana link.html i umesto *JavaScript* fajla crtajcvor.js, ona poziva posebno kreiran fajl crtajlink.js, koji sem funkcija ***DodajCvor()*** i ***CrtajCvor()*** sadrži još dve funkcije. Naime, ovom fajlu je potrebno dodati funkcije koje se pozivaju klikom na ***DodajLink()*** i ***CrtajLink()***, a to su:

```
function DodajLink () {
var x=document.getElementById("C1").value;
var y=document.getElementById("C2").value;
var p=document.getElementById("Cena").value;
var d=document.getElementById("Kasnjenje").value;
var c=document.getElementById("Kapacitet").value;
var NoviLink= new Link(x,y,p,c,d);
Linkovi.push(NoviLink);
}

function CrtajLink() {
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
CrtajLinkove(c,ctx,Linkovi);
}
```

Funkcija ***DodajLink()*** uzima vrednosti parametara linka iz polja za unos podataka i njih koristi kao parametre funkcije ***Link()***, koja kreira objekat *NoviLink*. Ovaj objekat se dodaje kao novi element nizu *Linkovi*, koji se u funkciji ***CrtajLink()*** koristi za crtanje linka između dva čvora. Fukcije ***Link()***, ***CrtajLinkove()*** i niz *Linkovi* su deo biblioteke lib.js.

4.3. Brisanje čvora, linka i toka

Za brisanje linka koristi se funkcija ***BrisiLink()***, koja je data sledećim kodom:

```
function BrisiLink(cvor1,cvor2,c,ctx) {
  var BrisiLinkInLinkovi=false;
  var ZaBrisanje;
  for (j=0;j<Linkovi.length;j++)  {
    var Linki=Linkovi[j];
    if (((Linki.PocetniCvor==cvor1) && (Linki.KrajnjiCvor==cvor2)) ||
((Linki.PocetniCvor==cvor2) && (Linki.KrajnjiCvor==cvor1))) {
      BrisiLinkInLinkovi=true;
      ZaBrisanje=j;
    }
  }

  if (((cvor1<0) || (cvor1>Cvorovi.length)) || ((cvor2<0) ||
(cvor2>Cvorovi.length)) || (cvor1==cvor2) || (BrisiLinkInLinkovi==false)) {
        alert("Proverite da li link koji želite da izbrišete postoji");
  }
  else {
    Linkovi.splice(ZaBrisanje,1);
    CrtajCvorove(c,ctx,Cvorovi);
    CrtajLinkove(c,ctx,Linkovi);
  }
}
```

Ulazni parametri ove funkcije su identifikacija početnog i završnog čvora linka i reference na kanvas element i njegov kontekst. Na početku funkcije zadaje se promenljiva *BrisiLinkInLinkovi* koja je setovana na vrednost "*false*". Pomoću ove promenljive proverava se da li stvarno postoji

link zadat preko korisničkog interfejsa kao link koji treba da se obriše. Ako ne postoji, nema ni svrhe brisati ga. Toj proveri upravo i služi FOR petlja koja sledi. U njoj se prolazi kroz sve linkove i za svaki element niza *Linkovi* se proverava da li su mu krajnji čvorovi oni čvorovi koje je korisnik zadao preko korisničkog interfejsa. Ako se pronađe takav link, pozicija u nizu *Linkovi* u kome se taj link nalazi pamti se u promenljivoj *ZaBrisanje*, a *BrisiLinkInLinkovi* se postavlja na "*true*". Nakon toga se validiraju uneti podaci. Naime, proverava se uslov da li je korisnik za čvorove uneo brojeve koji su manji od 0 ili koji su veći od dužine niza *Cvorovi* ili pak link između njih ne postoji. Ukoliko je to tačno, podaci su nevalidni i korisniku se ispisuje poruka "Proverite da li link koji želite da izbrišete postoji". Ukoliko uslov nije zadovoljen, odnosno, ukoliko su uneti podaci tačni, može se pristupiti brisanju linka. Pošto brisanje linka podrazumeva uklanjanje elementa niza sa određenim indeksom, konkretno indeksom *ZaBrisanje*, pogodno je iskoristiti *JavaScript* metodu ***splice()***. Sintaksa ove metode je *niz.splice(indeks, broj, vrednost1,...vrednostX)*. Parametar *indeks* je obavezan i to je celobrojna vrednost koja označava poziciju na kojoj treba dodati ili ukloniti neki element. Negativne vrednosti se koriste za specificiranje pozicije sa kraja niza. Parametar *broj* je broj elemenata koje treba ukloniti iz niza. Takođe je obavezan i ako ne treba ukloniti nijedan element, zadaje mu se vrednost 0. Parametri *vrednost1,...vrednostX* su opcioni i označavaju vrednosti novih elemenata koje treba dodati nizu. Metoda ***splice()*** vraća novi niz, sa uklonjenim ili dodatim elementima u odnosu na niz koji jos je prosleđen [7]. Na primer, dat je niz *planete=[Merkur, Venera, Neptun, Jupiter, Saturn]*. Naredba *planete.splice(2, 1, "Zemlja", "Mars");* će od pozicije dva ukloniti jedan element i dodati dva nova elementa. Kao rezultat dobiće se niz *[Merkur, Venera, Zemlja, Mars, Jupiter, Saturn]*. Primena ove metode za uklanjanje jednog elementa niza *Linkovi* na poziciji *ZaBrisanje* je naredba *Linkovi.splice(ZaBrisanje,1);*. Na ovaj način je dobijen niz *Linkovi*, sa istim redosledom elemenata, samo bez linka koji je korisnik zadao da treba da bude izbrisan. Poslednje dve funkcije vrše iscrtavanje čvorova i linkova, respektivno, kao ilustraciju da je link izbrisan iz topologije mreže.

Pored brisanja linka, potrebno je omogućiti i brisanje zadatog čvora. S obzirom da bi brisanjem čvora ostao link sa nepostojećim čvorom, potrebno je izbrisati i linkove koje taj čvor povezuje sa drugim čvorovima. Isto što važi za linkove važi i za tokove. Kako se redni brojevi čvorova smanjuju za jedan posle rednog broja obrisanog čvora, moraju se ažurirati i osobine *Linki.PocetniCvor* i *Linki.KrajnjiCvor*. Funkcija za brisanje čvorova naziva se ***BrisiCvor()***:

```
function BrisiCvor(x,c,ctx) {
  if ((x<1) || (x>Cvorovi.length)) {
    alert("Uneli ste pogresnu vrednost");
  }
  else {
    var BrisiCvorID=x-1;
    Cvorovi.splice(BrisiCvorID,1);
    CrtajCvorove(c,ctx,Cvorovi)

   var indeksi=[]; //članovi niza Linkovi koji ce se izbaciti iz niza jer sadrže čvor koji se uklanja
   for (j=0;j<Linkovi.length;j++)  {
    var Linki=Linkovi[j];
    if ((Linki.PocetniCvor==x) || (Linki.KrajnjiCvor==x)) {
      indeksi.push(j);
    }
   }
   var temp=[];
   for (j=0;j<Linkovi.length;j++)  {
    if  (indeksi.indexOf(j)==-1) {  var Linki=Linkovi[j];
      temp.push(Linki);
```

```
    }
   }
   Linkovi=temp;
   for (j=0;j<Linkovi.length;j++)  { //Ažuriranje niza Linkovi u cilju
ispravljanja identifikacija čvorova izmedju kojih se nalaze
    var Linki=Linkovi[j];
    if (Linki.PocetniCvor>x) {
      Linki.PocetniCvor=(Linki.PocetniCvor-1);
    }
    if (Linki.KrajnjiCvor>x) {
      Linki.KrajnjiCvor=(Linki.KrajnjiCvor-1);
    }
   }
  CrtajLinkove(c,ctx,Linkovi);

     var indeksi=[]; //članovi niza Linkovi koji ce se izbaciti iz niza jer
sadrže čvor koji se uklanja
   for (j=0;j<Tokovi.length;j++)  {
    var Toki=Tokovi[j];
    if ((Toki.PocetniCvorToka==x) || (Toki.ZavrsniCvorToka==x)) {
      indeksi.push(j);
    }
   }
   var temp=[];
   for (j=0;j<Tokovi.length;j++)  {
    if  (indeksi.indexOf(j)==-1) {  var Toki=Tokovi[j];
      temp.push(Toki);
    }
   }
   Tokovi=temp;
   for (j=0;j<Tokovi.length;j++)  { //Ažuriranje niza Linkovi u cilju
ispravljanja identifikacija čvorova izmedju kojih se nalaze
    var Toki=Tokovi[j];
    if (Toki.PocetniCvorToka>x) {
      Toki.PocetniCvorToka=(Toki.PocetniCvorToka-1);
    }
    if (Toki.ZavrsniCvorToka>x) {
      Toki.ZavrsniCvorToka=(Toki.ZavrsniCvorToka-1);
    }
   }

  }
}
```

Parametri koji se prosleđuju ovoj funkciji su redni broj čvora kojeg korisnik želi da izbriše i reference na *canvas* element i njegov kontekst. Parametar *x* je redni broj čvora, *c* je referenca na *canvas*, a *ctx* je kontekst elementa *canvas*. Funkcija ***BrisiCvor()*** prvo proverava da li je redni broj čvora manji od 1 ili veći od dužine niza *Cvorovi*. Ukoliko jeste, uneti redni broj je nevalidan i izbacuje se obaveštenje korisniku. Ukoliko uslov nije tačan, može se pristupiti brisanju čvora. S obzirom da korisnik u polje za unos podataka unosi redni broj čvora u mrežnoj topologiji, a indeksiranje nizova u *JavaScript* jeziku počinje od nule, da bi se dobio indeks elementa niza *Cvorovi* koji se treba izbrisati, uneti redni broj čvora za brisanje se mora umanjiti za jedan. Ova vrednost se pridružuje lokalnoj promenljivoj *BrisiCvorID*. Princip brisanja čvora je isti kao za brisanje linka, pa se i ovde primenjuje metoda ***splice()***. Preostali čvorovi se zatim iscrtavaju na topologiji pomoću funkcije ***CrtajCvorove***. Kako je svaki čvor vezan za barem jedan link, potrebno

je obrisati i odgovarajuće linkove. U tom cilju se definiše niz *indeksi*. U ovaj niz će biti smešteni oni indeksi niza *Linkovi* koji odgovaraju elementima čiji je početni ili krajnji čvor redni broj čvora koji se treba obrisati, a to je *x*. Elementi niza indeksi se određuju prolazom kroz elemente niza *Linkovi*, pomoću FOR petlje. U svakom prolazu ispituje se da li dati link ima kao početni ili krajnji čvor *x* i ako ima, indeks tog linka u nizu *Linkovi* se dodaje nizu indeksi. Dodavanje se obavlja pomoću metode ***push()***. Ova metoda dodaje nove elemente na kraj niza i njena sintaksa je *niz.push(vrednost1,...vrednostX)*, gde su parametri *vrednost1,...vrednostX* vrednosti elemenata koji se dodaju nizu. Dakle, dobijen je niz indeksa elemenata koje treba izbrisati iz niza *Linkovi*. Pošto je krajnji cilj tog brisanja niz *Linkovi*, samo bez elemenata sa indeksima koji predstavljaju elemente niza *indeksi*, ažuriran niz *Linkovi* se dobija građenjem novog niza kao u sledećem delu funkcije:

```
var temp=[];
for (j=0;j<Linkovi.length;j++)  {
if  (indeksi.indexOf(j)==-1) {  var Linki=Linkovi[j];
temp.push(Linki);
}
}
Linkovi=temp;
```

Kao što je prikazano, uvodi se novi niz *temp*. Prolazi se kroz niz *Linkovi* pomoću FOR petlje i u svakom prolazu ispituje se da li je vrednost indeksa *j* član niza *indeksi*. To se vrši pomoću metode ***indexOf()***. Ova metoda pretražuje zadatu vrednost u nizu i vraća njenu poziciju. Ukoliko nema rezultata pretrage, u ovom slučaju ukoliko se *j* ne nalazi u nizu *indeksi*, vraća -1. U tom slučaju će se element niza *Linkovi* sa indeksom *j* dodati na kraj niza *temp* i po izlasku iz FOR petlje, *temp* će predstavljati niz *Linkovi* bez linkova koji su povezani sa čvorom *x*. Potrebno je jos niz *temp* pridružiti globalnom nizu *Linkovi*. Na kraju samo preostaje ažurirati objekte linkova tako da pokazuju na adekvatne čvorove koje povezuju. To znači da, ako je link povezivao čvorove, na primer, 3 i 7, a čvor 5 je izbrisan, čvor 7 će sada postati čvor 6 pa će link povezivati čvorove 3 i 6. Sledeći kod realizuje ovaj zahtev:

```
for (j=0;j<Linkovi.length;j++)  { //Azuriranje niza Linkovi u cilju ispravljanja
identifikacija cvorova izmedju kojih se nalaze
    var Linki=Linkovi[j];
    if (Linki.PocetniCvor>x) {
      Linki.PocetniCvor=(Linki.PocetniCvor-1);
    }
    if (Linki.KrajnjiCvor>x) {
      Linki.KrajnjiCvor=(Linki.KrajnjiCvor-1);
    }
   }
```

Prolazeći kroz niz linkova, za svaki link se ispituje da li su mu početni i krajnji čvor veći od unetog rednog broja čvora. Ukoliko jesu, vrednosti ovih osobina se umanjuju za jedan. Preostali linkovi se crtaju, a nakon toga vrši se brisanje tokova koji sadrže čvor koji se briše, analogno brisanju linkova. Funkcija ***BrisanjeCvora()*** je time završena.

Funkcija za brisanje toka se naziva ***BrisiTok()*** i njeni ulazni podaci su početni cvor toka i krajnji čvor toka, odnosno *PocCvorToka* i *KrajCvorToka*, respektivno. Slično kao kod brisanja linka, uvodi se lokalna promenljiva za funkciju *BrisiTokInTokovi*, koja sadrži informaciju o tome da li su uneti brojevi čvorova zaista krajnje tačke toka podataka u topologiji. Inicijalna vrednost ove

promenljive je "*false*". Definiše se jos jedna promenljiva, *ZaBrisanje*, koja će se upotrebiti za skladištenje indeksa elementa u nizu *Tokovi* koga treba izbrisati. Kod funkcije ***BrisiTok()*** je sledeći:

```
function BrisiTok(PocCvorToka,KrajCvorToka) {
  var BrisiTokInTokovi=false;
  var ZaBrisanje;
  for (j=0;j<Tokovi.length;j++)  {
    var Toki=Tokovi[j];
    if (((Toki.PocetniCvorToka==PocCvorToka) &&
(Toki.ZavrsniCvorToka==KrajCvorToka)) || ((Toki.PocetniCvorToka==KrajCvorToka)
&& (Toki.ZavrsniCvorToka==PocCvorToka))) {
      BrisiTokInTokovi=true;
      ZaBrisanje=j;
    }
  }
  if (((PocCvorToka<0) || (PocCvorToka>Cvorovi.length)) || ((KrajCvorToka<0) ||
(KrajCvorToka>Cvorovi.length)) || (PocCvorToka==KrajCvorToka) ||
(BrisiTokInTokovi==false)) {
    alert("Proverite da li tok koji želite da izbrišete postoji");
  }
  else {
    Tokovi.splice(ZaBrisanje,1);
   }
}
```

Da bi se proverilo da li tok koji je zadat čvorovima *PocKrajToka* i *KrajCvorToka* zaista postoji u topologiji i da bi se odredio njegov položaj u nizu *Tokovi*, prolazi se kroz niz *Tokovi* uz pomoć FOR ciklusa. Uslov u ciklusu proverava da li su osobine *Toki.PocetniCvorToka* i *Toki.ZavrsniCvorToka* elementa niza *Tokovi* sa indeksom *j* jednake zadatim čvorovima tokova i ukoliko jesu, *BrisiTokInTokovi* se postavlja na vrednost "*true*", a promenljivoj *ZaBrisanje* se dodeljuje vrednost indeksa *j*. Nakon FOR ciklusa proverava se uslov da li su vrednosti unetih čvorova veće od nule, manje od dužine niza *Cvorovi* i da li željeni tok za brisanje postoji. Ukoliko uslov nije zadovoljen, korisnik se obaveštava o grešci, a ukoliko jeste, tok se briše pomoću metode ***splice()***, naredbom `Tokovi.splice(ZaBrisanje,1);`. To znači da iz niza *Tokovi* treba ukloniti jedan element počevši od indeksa *ZaBrisanje*.

4.4. Ažuriranje čvora, linka i toka

Za ažuriranje čvora koristi se funkcija ***AzurirajNode()***. Ulazni parametri ove funkcije su redni broj čvora koji se ažurira, *x*, i objekat *NoviCvor*, koji predstavlja čvor sa izmenjenim željenim parametrima. Funkcija je sledeća:

```
function AzurirajNode(x,NoviCvor) {
  x=x-1;
  Cvorovi[x]=NoviCvor;
}
```

Promenljiva *x* se umanjuje za jedan zato što predstavlja redni broj čvora sa aspekta korisnika, a indeksiranje nizova u *Javascript* jeziku počinje od nule. Na taj način promenljiva *x* postaje jednaka vrednosti indeksa tog čvora, koji se ažurira, u nizu *Cvorovi*. U drugoj naredbi se elementu sa tim indeksom u nizu objekata *Cvorovi* dodeljuje objekat *NoviCvor*, koji sadrži ažurirane osobine objekta čvora.

Da bi se ažurirao link u nizu *Linkovi*, neophodne su informacije o krajnjim čvorovima linka koji se ažurira, kao i ažurirani parametri. Ove parametre treba proslediti funkciji ***AzurirajLink()***. Ažurirani parametri jednog linka se prosleđuju putem objekta *NoviLink*, radi ažuriranja niza *Linkovi*. Funkcija ***AzurirajLink()*** je sledeća:

```
function AzurirajLink (updateLinkNodeA,updateLinkNodeB,NoviLink) {
  for (j=0;j<Linkovi.length;j++)  {
    var Linki=Linkovi[j];
    if (((Linki.PocetniCvor==updateLinkNodeA) ||
(Linki.KrajnjiCvor==updateLinkNodeA)) && ((Linki.PocetniCvor==updateLinkNodeB)
|| (Linki.KrajnjiCvor==updateLinkNodeB))) {
      Linkovi[j]=NoviLink;
    }
  }
}
```

Funkcija započinje FOR ciklusom, koji proverava da li *j*-ti element niza *Linkovi* ima početni čvor i krajnji, ili krajnji i početni, čvor (pošto su linkovi dvosmerni), jednak onima koje je zadao korisnik, a koji su prosleđeni funkciji preko parametara *updateLinkNodeA* i *updateLinkNodeB*. Ukoliko je taj uslov tačan, *j*-ti element niza *Linkovi* se postavlja na vrednost *NoviLink*. Na taj način je izvršeno ažuriranje niza *Linkovi*.

Osim ažuriranja čvora i linka, moguće je ažurirati i tok. S obzirom da je tok, kao i link, definisan identifikacijama dva čvora, funkciji kojom se ažurira niz tokova, potrebno je proslediti početni i krajnji čvor toka, da bi se tok pronašao u nizu *Tokovi*, kao i novu vrednost koju treba dodeliti datom toku. Početni kraj toka je definisan parametrom *updateFlowNodeA*, krajnji čvor parametrom *updateFlowNodeB*, a novu vrednost toka predstavlja parametar *NoviTok*. Referenca na *canvas* i kontekst se ne prosleđuju zato što se predstavljanje tokova, u ovom slučaju, ne realizuje u elementu *canvas*. Funkcija ***AzurirajTok()*** je predstavljena sledećim kodom:

```
function AzurirajTok(updateFlowNodeA,updateFlowNodeB,NoviTok) {
  for (j=0;j<Tokovi.length;j++)  {
    var Toki=Tokovi[j];
      if (((Toki.PocetniCvorToka==updateFlowNodeA) ||
(Toki.ZavrsniCvorToka==updateFlowNodeA)) &&
((Toki.PocetniCvorToka==updateFlowNodeB) ||
(Toki.ZavrsniCvorToka==updateFlowNodeB))) {
        Tokovi[j]=NoviTok;
      }
  }
}
```

Kao i za link, funkcija počinje FOR ciklusom koji proverava da li *j*-ti element niza *Tokovi* za početni i završni, ili završni i početni, čvor toka ima *updateFlowNodeA* i *updateFlowNodeB* i ukoliko ima, dodeljuje mu vrednost objekta *NoviTok*, sa ažuriranim osobinama toka.

4.5. Validacija parametara čvora, linka i toka

Nakon što korisnik preko korisničkog interfejsa unese vrednosti za parametre čvora, linka i toka, a pre nego što se parametri dodaju u nizove Cvorovi, Linkovi i Tokovi, potrebno je proveriti njihovu ispravnost, kako bi se omogućio korektan rad aplikacije. Funkcije za proveravanje tačnosti unetih parametara kao ulazne parametre imaju parametre čvora, linka ili toka, a kao izlazni parametar vrednost *boolean* promenljive *valid*.

4.5.1. Validacija parametara čvora

Funkcija za validaciju parametara čvora naziva se ***ValidateNodeParameters()***. Njeni ulazni parametri su kontekst elementa *canvas* i parametri čvora - njegove koordinate i kašnjenje, a izlazni parametar je *valid*, koji ima vrednost "true", ako su parametri ispravni, odnosno "false", u suprotnom slučaju. Kod funkcije ***ValidateNodeParameters()*** je sledeći:

```
function ValidateNodeParameters(c,x,y,d) {  //c je kontekst, x, y i d su
koordinate cvora i kašnjenje čvora
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
  if ( (x.match(patt1)) || (y.match(patt1)) || (d.match(patt2)) || (x==null ||
x=="") || (y==null || y=="") || (d==null || d=="")) {
  //ako parametri sadrze slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Parametri ne smeju sadržati slova, razmake ili specijalne
karaktere!');
  }

  x=parseFloat(x);
  y=parseFloat(y);
  if (((x>c.width) || (x<0)) || ((y>c.height) || (y<0))) {
  //ako tačka izlazi iz okvira canvas-a
        valid=false;
        alert ('Uneli ste pogrešne koordinate!');
  }
  if (Cvorovi.length>0) {
    //ako je unet bar jedan čvor
    T=Cvorovi.length;
    for (i=0;i<T;i++){
      var Cvori=Cvorovi[i];
      if (Cvori.KoordinataX==x) {
        if (Cvori.KoordinataY==y) {
          //vec postoji čvor sa tim koordinatama u nizu Cvorovi
          valid=false;
          alert ('Dati čvor vec postoji!');
        }
      }
    }
   }
  return valid;
}
```

Promenljiva *valid* se na početku postavlja na vrednost "*true*". Prvo što treba proveriti jeste da li uneti podaci sadrže slova ili specijalne karaktere. U tom cilju se deklarišu uzorci *patt1 i patt2*, kao jedno pojavljivanje slova ili specijalnog karaktera, s tim što *patt2* obuhvata i pojavljivanje tačke. Atribut *i* na kraju uzorka znači da je pretraga neosetljiva na veličinu slova. Pomoću metode *match(),* u uslovu koji sledi, proverava se da li među unetim parametrima postoji slovo ili specijalan karakter. Kako su koordinate tipa *float*, parametri *x* i *y* mogu da sadrže tačku. U uslovu se takođe proverava da li je neki od parametara prosleđen bez vrednosti. Ukoliko jeste, *valid* se postavlja na vrednost "*false*" i korisnik se obaveštava o grešci porukom "Podaci ne smeju sadržati slova, razmake ili specijalne karaktere!". Ukoliko je korisnik uneo numeričke podatke za koordinate čvora, proverava se da li uneti čvor izlazi iz okvira elementa *canvas*, u kojem slučaju se *valid*

postavlja na vrednost "*false*" i korisnik se obaveštava o grešci. Isto se dešava i u slučaju da čvor sa unetim koordinatama već postoji u nizu *Cvorovi* i stoga nema smisla dodati još jedan.

Parametri čvora se unose i prilikom ažuriranja čvora, s tom razlikom da je potrebno da korisnik unese i redni broj čvora koji se ažurira. Funkcija za validaciju podataka za ažuriranje čvora je ***ValidateUpdateNodeParameters()***:

```
function ValidateUpdateNodeParameters(num,c,x,y,d) {
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  if ( (num.match(patt1)) || (x.match(patt2)) || (y.match(patt2)) ||
(d.match(patt1)) || (num==null || num=="") || (x==null || x=="") || (y==null ||
y=="") || (d==null || d=="")) {
  //ako podaci sadrze slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Podaci ne smeju sadržati slova, razmake ili specijalne karaktere!');
  }
  num=parseInt(num);
  x=parseFloat(x);
  y=parseFloat(y);
  if ((num<1) || (num>Cvorovi.length)) {
  valid=false;
  alert ('Proverite da li cvor koji azurirate postoji!');
  }
  if (((x>c.width) || (x<0)) || ((y>c.height) || (y<0))) {
  //ako tacka izlazi iz okvira canvasa
    valid=false;
    alert ('Uneli ste pogresne koordinate!');
  }
  if (Cvorovi.length>0) {
    //ako je unet bar jedan čvor
    T=Cvorovi.length;
    for (i=0;i<T;i++) {
      var Cvori=Cvorovi[i];
      if (Cvori.KoordinataX==x) {
        if (Cvori.KoordinataY==y) {
          //vec postoji čvor sa tim koordinatama u nizu Cvorovi
          valid=false;
          alert ('Dati čvor vec postoji!');
        }
      }
    }
  }
  return valid;
}
```

Ulazni parametar *num* je redni broj čvora sa aspekta korisnika, *c* je kontekst *canvas*-a, *x* i *y* su koordinate, a *d* je kašnjenje čvora. Na početku funkcije se deklarišu promenljive *valid*, koja je izlazni parametar funkcije, i *patt1* i *patt2*, koje su uzoci slova i specijalnih karaktera. Uzorak *patt1* obuhvata i pojavljivanje tačke, koje ne sme biti u *integer* tipu podataka, u ovom slučaju promenljivama *num* i *d*. U uslovu koji sledi proverava se da li se uzorak *patt1*, odnosno *patt2* nalazi među ulaznim parametrima i da li su svi ulazni parametri uneti. Ukoliko bilo koji parametar sadrži bar jedno slovo ili specijalan karakter ili je bilo koji od parametara prosleđen bez vrednosit, *valid* se postavlja na "*false*" i korisnik se obaveštava o neispravnosi unetih podataka. Time je obezbeđen unos numeričkih podataka, međutim, potrebno je da ti numerički podaci budu i smisleni. Tako su

pokriveni slučajevi da korisnik za redni broj čvora unese broj manji od jedinice ili broj veći od ukupnog broja čvorova, slučaj kada nove koordinate određuju tačku van prostora *canvas* elementa i slučaj kada čvor zadatim novim koordinatama već postoji u topologiji, odnosno u nizu *Cvorovi*. Za sva tri slučaja *valid* se postavlja na "*false*", a korisniku se prikazuje adekvatna poruka o grešci.

4.5.2. Validacija parametara linka

Proveru ispravnosti unetih parametara linka vrši funkcija ***ValidateLinkParameters()***. Ulazni parametri ove funkcije su pocetni čvor linka *x*, krajnji čvor linka *y*, cena linka *c*, kapacitet linka *C* i kašnjenje *d*. Izlazni parametar je *valid*, koji ima vrednost "*true*", ako su ispunjeni zadati uslovi validnosti. Funkcija je data sledećim kodom:

```
function ValidateLinkParameters(x,y,c,C,d) {
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  if ( (x.match(patt1)) || (y.match(patt1)) || (c.match(patt1)) ||
(C.match(patt2)) || (d.match(patt1)) || (x==null || x=="") || (y==null || y=="")
|| (c==null || c=="") || (C==null || C=="") || (d==null || d=="") ) {
    //ako parametri sadrže slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Podaci ne smeju da sadrže slova, razmake ili specijalne
karaktere!');
  }
  if (x==y) {
    //unet je isti čvor za početni i krajnji čvor
    valid=false;
    alert ('Čvorovi moraju biti različiti!');
  }

  M=Cvorovi.length;
  N=Linkovi.length;
  if ((x<1) || (y<1) || (x>M) || (y>M)) {
    //ako čvor sa unetim rednim brojem još uvek ne postoji
    valid=false;
    alert ('Proverite da li uneti čvor postoji!');
  }

  for (i=0;i<N;i++) {
    var Linki=Linkovi[i];
    if (Linki.PocetniCvor==x) {
      if (Linki.KrajnjiCvor==y) {
        //ako novi link već postoji u nizu Linkovi
        valid=false;
        alert ('Dati link vec postoji!');
      }
    }
    if (Linki.KrajnjiCvor==x) {
      //ako novi tok već postoji u nizu Linkovi
      if (Linki.PocetniCvor==y) {
      valid=false;
      alert ('Dati link vec postoji!');
      }
    }
   }
  return valid;
}
```

Kako svi ulazni parametri treba da su brojevi, pomoću uzoraka *patt* i *patt2* se proverava da li parametri sadrže slova ili specijalne karaktere. Uzorak *patt2* ne obuhvata pojavljivanje tačke, koja se može pojavljivati u *float* tipu podataka, u ovom slučaju promenljivoj *C*. Takođe, obavezno je uneti svaki parametar. Početni i krajnji čvor, odnosno promenljive *x* i *y* ne smeju imati istu vrednost i moraju uzimati vrednosti od 1 do broja čvorova u topologiji, odnosno broja elemenata u nizu *Cvorovi*. Promenljiva koja označava broj čvorova je *M*. Na kraju funkcije se proverava da li link između čvorova *x* i *y* već postoji u nizu *Linkovi*, čime se sprečava dodavanje jednog linka više puta. Ukoliko bar jedan od navedenih uslova nije tačan, funkcija vraća vrednost "*false*" i korisnik se obaveštava porukom o tome gde je pogrešio.

Kada ažurira link, korisnik unosi početni i krajnji čvor željenog linka, za koji određuje novu cenu, novi kapacitet ili novo kašnjenje. Svi parametri su obavezni, s tim što se ne mora uneti nova vrednost i za cenu i za kapacitet i za kašnjenje. Funkcija koja proverava ispravnost unetih podataka prilikom ažuriranja linka je ***ValidateUpdateLinkParameters()***, data sledećim kodom:

```
function ValidateUpdateLinkParameters(NodeA,NodeB,c,C,d) {
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  if ( (NodeA.match(patt1)) || (NodeB.match(patt1)) || (c.match(patt1)) ||
(C.match(patt2)) || (d.match(patt1)) || (NodeA==null || NodeA=="") ||
(NodeB==null || NodeB=="") || (c==null || c=="") || (C==null || C=="") ||
(d==null || d=="") ) {
    //ako parametri sadrže slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Podaci ne smeju da sadrže slova, razmake ili specijalne
karaktere!');
  }
  if (NodeA==NodeB) {
  //unet je isti čvor za početni i krajnji čvor
  valid=false;
  alert ('Čvorovi moraju biti različiti!');
  }
  NodeA=parseInt(NodeA);
  NodeB=parseInt(NodeB);
  M=Cvorovi.length;
  N=Linkovi.length;
  var LinkIsNotInLinkovi=true;   //Link koji se ažurira nije u nizu Linkovi
  for (i=0;i<N;i++) {      //Prolazi se kroz nizu Linkovi
    var Linki=Linkovi[i];
    if ((Linki.PocetniCvor==NodeA) && (Linki.KrajnjiCvor==NodeB))  {
      //Pronađen je link sa zadatim čvorovima
      LinkIsNotInLinkovi=false;
      //Link koji se ažurira je u nizu Linkovi
    }
    if ((Linki.KrajnjiCvor==NodeA) && (Linki.PocetniCvor==NodeB))  {
      LinkIsNotInLinkovi=false;
    }
  }
  if (LinkIsNotInLinkovi) {   //Link koji se ažurira nije u nizu Linkovi
    valid=false;
    alert ('Proverite da li link koji ažurirate postoji!');
  }
  return valid;
}
```

Promenljiva koja označava početni čvor linka je *NodeA*, promenljiva *NodeB* je krajnji čvor linka. Promenljive *c*, *C* i *d* su nova cena, novi kapacitet i novo kašnjenje linka, respektivno. Pošto su svi parametri numeričke vrednosti, deklariše su uzorci *patt1 i patt2*, koji pomoću metode ***match()*** proveravaju da li se među parametrima nalazi slovo ili specijalan karakter, na isti način kao kod dodavanja novog linka. U istom uslovu proverava se da li je neko od polja forme prosleđeno nepopunjeno. Tačna vrednost uslova znači da uneti podaci za ažuriranje linka nisu ispravni jer su potencijalni izvori greške zbog čega se korisnik obaveštava porukom "Podaci ne smeju da sadrže slova, razmake ili specijalne karaktere!", a izlazni parametar funkcije, *valid*, dobija vrednost "*false*". Isto kao i kod dodavanja novog linka, početni i krajnji čvor moraju da budu različiti i link sa početnim i krajnjim čvorom mora postojati u nizu *Linkovi*, da bi se uopšte mogao ažurirati. Ukoliko su ti uslovi nezadovoljeni, korisniku se ispisuje odgovarajuća poruka i promenljiva *valid* se postavlja na na vrednost "*false*".

4.5.3. Validacija parametara toka

Parametri toka su početni i krajnji čvor toka i kapacitet toka. Funkcija koja proverava ispravnost vrednosti ovih parametara je ***ValidateFlowParameters()*** i data je sledećim kodom:

```
function ValidateFlowParameters(x,y,C) {
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:."';'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  if ( (x.match(patt1)) || (y.match(patt1)) || (C.match(patt2)) || (x==null ||
x=="") || (y==null || y=="") || (C==null || C=="") ) {
  //ako parametri sadrze slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Podaci ne smeju da sadrže slova, razmake ili specijalne
karaktere!');
  }
  if (x==y) {
  //unet je isti čvor za početni i krajnji čvor
  valid=false;
  alert ('Čvorovi moraju biti različiti!');
  }

  x=parseInt(x);
  y=parseInt(y);
  M=Cvorovi.length;
  N=Tokovi.length;
  if ((x<1) || (y<1) || (x>M) || (y>M)) {
    //ako čvor sa unetim rednim brojem još uvek ne postoji
    valid=false;
    alert ('Proverite da li uneti čvor postoji!');
  }
  for (i=0;i<N;i++) {
    var Toki=Tokovi[i];
    if (Toki.PocetniCvorToka==x) {
      if (Toki.ZavrsniCvorToka==y) {
        //ako novi tok već postoji u nizu Tokovi
        valid=false;
        alert ('Dati tok vec postoji!');
      }
    }
    if (Toki.ZavrsniCvorToka==x) {
      if (Toki.PocetniCvorToka==y) {
        //ako novi tok već postoji u nizu Tokovi
```

```
        valid=false;
        alert ('Dati tok vec postoji!');
      }
    }
  }
  return valid;
}
```

Ulazni parametar ove funkcije *x* označava početni čvor, parametar *y* je krajnji čvor, parametar *C* je kapacitet toka. Prva promenljiva koja se deklariše u telu funkcije je *valid.* Ona je izlazni parametar funkcije i ima vrednost "*true*", ako je korisnik uneo ispravne podatke, odnosno "*false*" u suprotnom slučaju. Zatim se deklarišu uzorci *patt1* i *patt2*, čija je svrha u proveri prisustva slova ili specijalnih karaktera u ulaznim parametrima funkcije pomoću funkcije *match()*. Razlika ova dva uzorka je u tome što *patt2* ne proverava pojavljivanje i tačke, koju korisnik može uneti u polje za kapacitet toka, jer je kapacitet promenljiva tipa *float*. U prvom uslovu se proverava da li slova i specijalni karakteri postoje u ulaznim parametrima i da li je neki parametar prosleđen bez unete vrednosti i ukoliko je uslov ispunjen, vrednost promenljive *valid* se postavlja na "*false*", a korisniku će da se prikaže poruka "Podaci ne smeju da sadrže slova, razmake i specijalne karaktere." Nakon toga sledi uslov u kome se proverava da li je za početni i krajnji čvor uneta ista vrednost, pa uslov u kome se proverava da li su redni brojevi unetih čvorova izvan opsega od jedan do dužine niza *Cvorovi* i na kraju sledi provera da li novi tok koji je definisan krajnjim čvorovima *x* i *y* već postoji u nizu Tokovi. Ukoliko je ispunjen, svaki od ova tri uslova, ekvivalentno prvom uslovu, dovodi to toga da se vrednost promenljive *valid* postavlja na "*false*", a korisniku se ispisuje odgovarajuća poruka.

Prilikom ažuriranja parametara toka, korisnik zadaje početni i krajnji čvor toka koji se ažurira, pomoću kojih će se identifikovati zadati tok u nizu *Tokovi*, i unosi novu vrednost kapaciteta. Stoga su to ulazni parametri funkcije za ažuriranje parametara toka, ***ValidateUpdateFlowParameters()***, koja je predstavljena sledećim kodom:

```
function ValidateUpdateFlowParameters(NodeA,NodeB,C) {
  var valid=true;
  var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
  var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
  if ( (NodeA.match(patt1)) || (NodeB.match(patt1)) || (C.match(patt2))  ||
(NodeA==null || NodeA=="") || (NodeB==null || NodeB=="") || (C==null || C=="") )
{
    //ako parametri sadrze slova, razmak i specijalne karaktere ili su prazni
    valid=false;
    alert ('Podaci ne smeju da sadrže slova, razmake ili specijalne
karaktere!');
  }
  if (NodeA==NodeB) {
    //unet je isti čvor za početni i krajnji čvor
    valid=false;
    alert ('Čvorovi moraju biti razliciti!');
  }
  NodeA=parseFloat(NodeA);
  NodeB=parseFloat(NodeB);
  M=Cvorovi.length;
  N=Tokovi.length;
  var FlowIsNotInTokovi=true;        //Tok koji se ažurira nije u nizu Tokovi
  for (i=0;i<N;i++) {                //Prolazi se kroz niz Tokovi
    var Toki=Tokovi[i];
    if ((Toki.PocetniCvorToka==NodeA) && (Toki.ZavrsniCvorToka==NodeB))  {
      //Pronađen je tok sa zadatim čvorovima
```

```
      FlowIsNotInTokovi=false;
      //Tok koji se ažurira je u nizu Tokovi
    }
    if ((Toki.ZavrsniCvorToka==NodeA) && (Toki.PocetniCvorToka==NodeB))  {
      FlowIsNotInTokovi=false;
    }
  }
  if (FlowIsNotInTokovi) {           //Tok koji se ažurira nije u nizu Tokovi
    valid=false;
    alert ('Proverite da li tok koji ažurirate postoji!');
  }
  return valid;
}
```

Ulazni parametar funkcije *NodeA* je početni čvor toka, *NodeB* je krajnji čvor toka, a *C* je nova vrednost kapaciteta. Na isti način kao kod dodavanja novog toka, prvo se proverava da li su svi parametri uneti i to bez slova i specijalnih karaktera. Drugi uslov proverava da li je za početni i krajnji čvor uneta različita vrednost, a treći uslov da li tok koji je zadat čvorovima *NodeA* i *NodeB* i koji se ažurira uopšte postoji u nizu Tokovi. Za svaki od navedenih uslova, ukoliko nije zadovoljen, promenljiva *valid* se postavlja na vrednost "*false*", a korisniku se ispisuje poruka o grešci koju je načinio prilikom unosa. Funkcija ***ValidateUpdateFlowParameters()*** na kraju vraća vrednost promenljive *valid*, signalizirajući da li su uneti parametri ispravno uneti ili ne i da li se niz Tokovi može ili ne može ažurirati shodno novim vrednostima koje je korisnik uneo.

5. PRIMERI PRIMENE BIBLIOTEKE FUNKCIJA

U ovom poglavlju će biti prikazani primeri primene funkcija opisanih u prethodnom poglavlju, a to su ručno unošenje i crtanje topologije mreže, primena Dijkstra algoritma na selektovani čvor i rad sa bazom podataka – upisivanje kreirane topologije u bazu, ispisivanje iz baze i brisanje postojeće topologije.

5.1. Ručno unošenje i crtanje topologije mreže

Radi lakše manipulacije *JavaScript* funkcijama i preglednijeg izgleda HTML stranice, za demonstraciju primene funkcija definisanih u poglavlju tri, biće korišćen kod *JavaScript* biblioteke *jQuery*, koja je preuzeta sa sajta [8]. Ova biblioteka pojednostavljuje pisanje skripti, olakšava navigaciju kroz dokument i selektovanje DOM elemenata, rad sa *Event* obektom i kreiranje animacija. Više detalja o *jQuery* biblioteci se može naći na [9].

Na HTML stranici za demonstraciju ručnog unošenja i crtanja topologije, trebalo bi prikazati polja za unošenje parametara čvora, linka i toka, zatim prikazati sve te unete parametre, *canvas* na kome će se crtati čvorovi i linkovi, kao i dugmad za crtanje čvora i linka. U primeru se definiše element <*canvas*> sa širinom 500px i 400px, sivog okvira širine 1px i sa apsolutnom pozicijom:

```
<canvas id="MojCanvas" width="500" height="400" style="border:1px solid #d3d3d3;
position:absolute; left:600px; top: 30px; right:30px;"></canvas>
```

Prvo što u *JavaScript* fajlu sa primerima treba uraditi jeste referencirati element *canvas* i naći njegov kontekst, kao preduslov za sve operacije vezane za *canvas*.

```
var c=document.getElementById("MojCanvas");
var ctx=c.getContext("2d");
var w,h;
```

Kako su parametri čvora koordinate centra kružnice na *canvas*-u, zgodno je imati prikaz koordinata tačke na *canvas*-u. To se može dobiti na osnovu koordinata miša. Da bi se dobile koordinate miša relativne u odnosu na *canvas*, kreirana je funkcija ***KoordinateMisa()***, koja vraća koordinate miša na osnovu koordinata u odnosu na trenutni prozor i pozicije *canvas*-a dobijene pomoću ***getBoundingClientRect()*** metode. Koordinate miša u odnosu na prozor dobijaju se pomoću *clientX* i *clientY* atributa objekta događaja koji je vezan za miša. *ClientX* je horizontalna, a *clientY* vertikalna koordinata miša u odnosu na prozor u trenutku kada je događaj izazvan. Događaj koji će se koristiti za pozivanje ove funkcije je pomeranje miša. Metoda ***getBoundingClientRect()*** vraća objekat sa osobinama "*top*", "*left*", "*right*", "*bottom*", odnosno koordinate gornjeg levog i donjeg desnog ugla pravougaonika koji ograđuje HTML element, u ovom slučaju *canvas*, stoga, da bi se dobile koordinate tačke na *canvas*-u, potrebno je oduzeti horizontalnu koordinatu levog gornjeg ugla *canvas*-a od horizontalne koordinate pozicije miša u prozoru i vertikalnu koordinatu levog gornjeg ugla *canvas*-a od vertikalne koordinate pozicije miša u prozoru. Funkcija ***KoordinateMisa()*** vraća *x* i *y* koordinatu tačke u *canvas*-u:

```
function KoordinateMisa(canvas, dogadjaj) {
  var rect = canvas.getBoundingClientRect();
  return {
    x: dogadjaj.clientX - rect.left,
    y: dogadjaj.clientY - rect.top
   };
 }
```

Kao što je spomenuto, ova funkcija će se pozivati na pomeranje miša i za to je upotrebljena metoda ***element.addEventListener()***, gde je element HTML element za koji se funkcija poziva. Sintaksa ove metode je ***element.addEventListener(događaj, funkcija, propagacija_događaja)***. Prvi parametar je tip događaja, tipa "*click*" ili "*mousemove*", drugi parametar je funkcija koja se poziva na događaj, a treći parametar je opcioni i predstavlja *boolean* vrednost, koja specificira koju propagaciju događaja koristiti. Ako je "*true*", propagacija je "*capturing*", u suprotnom je "*bubbling*". Propagacija događaja definiše kako će se izvršiti redosled događaja za dva HTML elementa, ako je jedan unutar drugog. Na primer, ako je <*p*> unutar <*div*> elementa i korisnik klikne na <*p*>, u "*bubbling*" propagaciji će se prvo izvršiti funkcija vezana za klik na <*p*>, pa tek onda funkcija vezana za klik na <*div*> element. U "*capturing*" propagaciji će se prvo izvršiti funkcija vezana za spoljašnji element, a zatim za unutrašnji, u ovom slučaju, prvo funkcija vezana za klik na <*div*>, pa funkcija vezana na klik na <*p*> element [9]. U metodi čiji je kod prikazan ispod, elementu *canvas* je na događaj pomeranja miša, dodeljena funkcija koja poziva funkciju koja vraća koordinate miša.

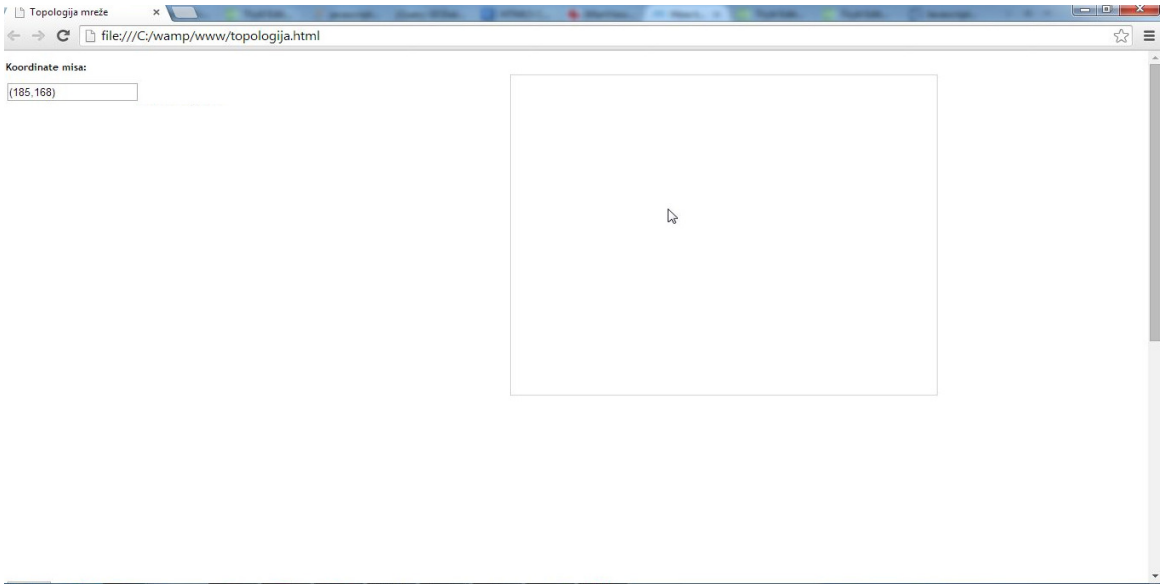
```
c.addEventListener('mousemove', function(event) {
  var PozicijaMisa = KoordinateMisa(c, event);
  var koordinate=document.getElementById("koordinate");
  koordinate.value= '(' + PozicijaMisa.x + ',' + PozicijaMisa.y + ')';
}, false);
```

Koordinate miša se zatim ispisuju u teksualnom polju čiji je *id="koordinate"*, kao na slici 5.1.1.

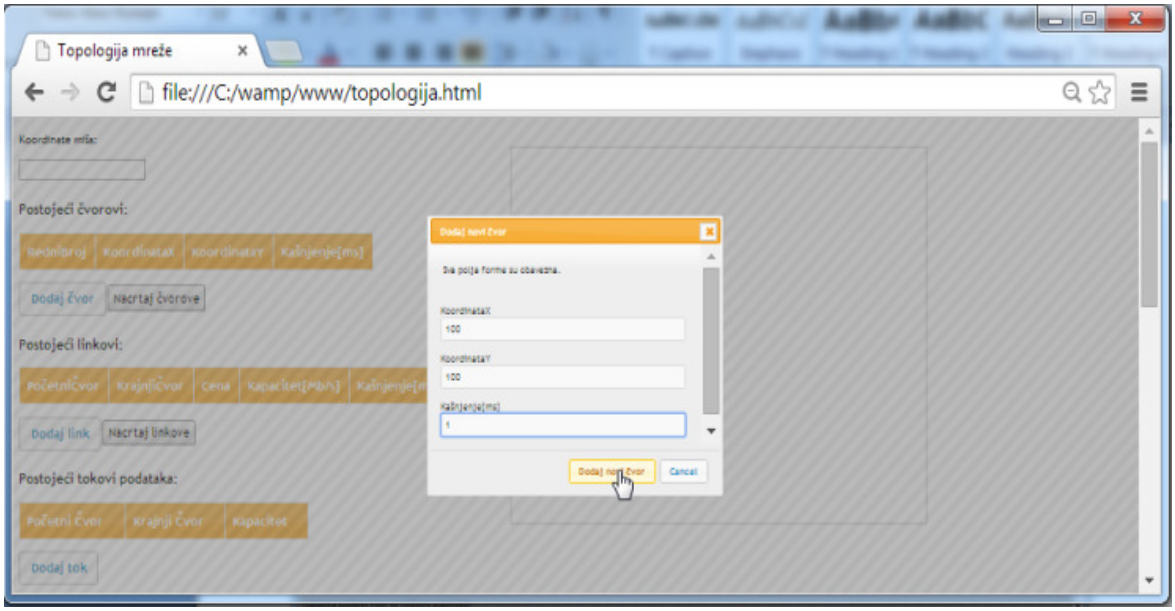
Takođe, potrebno je obezbediti unos parametara čvora, linka i toka i njihovo ispisivanje na HTML stranici. Kako treba uneti više čvorova, linkova i tokova i svaki od ovih elemenata ima barem tri parametra, radi preglednijeg izgleda HTML stranice koristiće se *jQuery* dijalog dodatak [10]. Naime, kad korisnik klikne na odgovarajuće dugme da bi dodao čvor/link/tok, otvoriće se prozor u kome se zahteva unos parametara datog elementa, kao na slici 5.1.2 za unos parametara čvora, i nakon uspešnog unosa, uneti parametri elementa će se prikazati u tabeli na HTML stranici, kao što je prikazano na slici 5.1.3. HTML kod dijaloga prikazanog na slici 5.1.2 je sledeći:

```
<div id="dialog-node" title="Dodaj novi čvor">
    <p class="validateTips">Sva polja forme su obavezna.</p>
    <form>
        <fieldset>
            <label for="KoordinataX">KoordinataX</label>
            <input type="text" name="KoordinataX" id="KoordinataX" class="text
ui-widget-content ui-corner-all" required>
            <label for="KoordinataY">KoordinataY</label>
            <input type="text" name="KoordinataY" id="KoordinataY" value=""
class="text ui-widget-content ui-corner-all" required>
            <label for="KasnjenjeCvora">Kašnjenje[ms]</label>
            <input type="text" name="KasnjenjeCvora" id="KasnjenjeCvora"
value="" class="text ui-widget-content ui-corner-all" required>
        </fieldset>
```

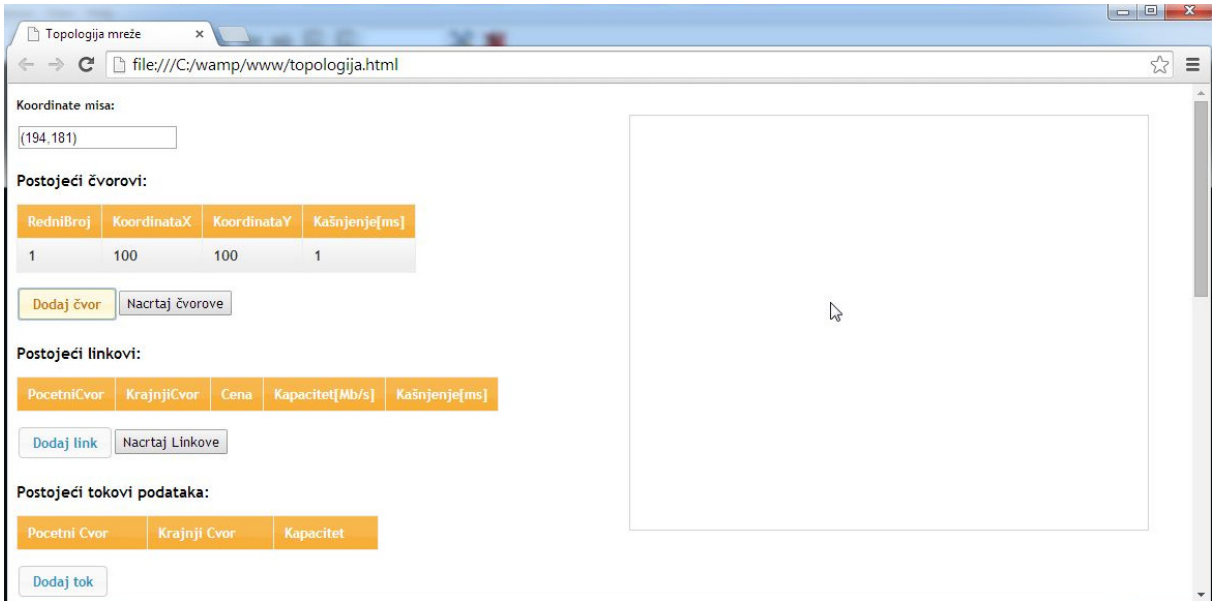
```
    </form>
</div>
```



Slika 5.1.1. Koordinate miša u *canvas*-u



Slika 5.1.2. Dodavanje novog čvora



Slika 5.1.3. Novi čvor prikazan u tabeli postojećih čvorova

Element <*fieldset*> grupiše elemente u formi i crta okvir oko te grupe elemenata, a element <*label*> definiše tekst koji će biti prikazan uz odgovarajući <*input*> element. Atribut *for*, elementa <*label*>, treba da bude jednak *id* atributu vezanog elementa kako bi se zajedno pretraživali. Definisanje promenljivih za skladištenje ulaznih podataka određenih poljima za unos podataka vrši se sledećom naredbom:

```
var KoordinataX = $( "#KoordinataX" ),
  KoordinataY = $( "#KoordinataY" ),
  KasnjenjeCvora = $( "#KasnjenjeCvora" ),
  allFieldsNode = $( [] ).add( KoordinataX ).add( KoordinataY ).add(
KasnjenjeCvora );
```

Tabela na slici 5.1.3 je kreirana HTML kodom:

```
<div id="nodes-contain" class="ui-widget">
        <h1>Postojeći čvorovi:</h1>
        <table id="nodes" class="ui-widget ui-widget-content">
            <thead>
            <tr class="ui-widget-header ">
            <th>RedniBroj</th>
            <th>KoordinataX</th>
            <th>KoordinataY</th>
            <th>Kašnjenje[ms]</th>
            </tr>
            </thead>
        <tbody>
        </tbody>
        </table>
        <button id="add-node">Dodaj čvor</button
</div>
```

Nakon što korisnik klikne na dugme "Dodaj čvor", čiji je *id="add-node"*, poziva se funkcija u kojoj se otvara opisani prozor za unošenje parametara čvora, što izvršava sledeća *jQuery* funkcija:

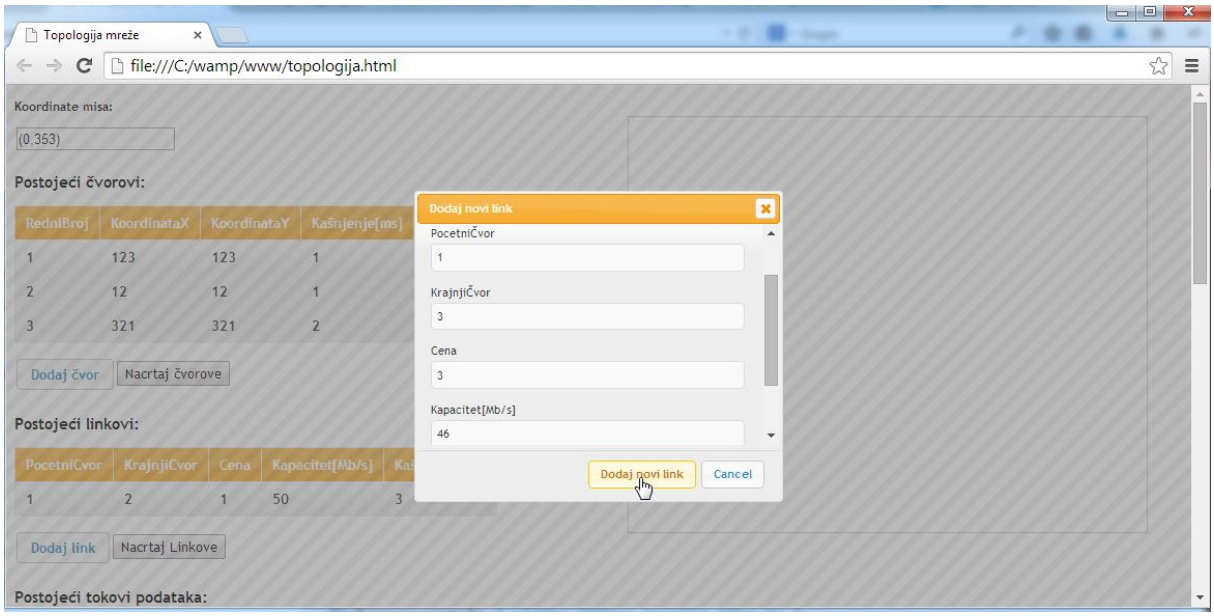
```
  $( "#add-node" ).button().click( function() {
      $( "#dialog-node" ).dialog( "open" );
  })
```

Prethodno je potrebno definisati veličinu, osobine i sadržaj dijaloga, prozora prikazanog na slici 5.1.2.

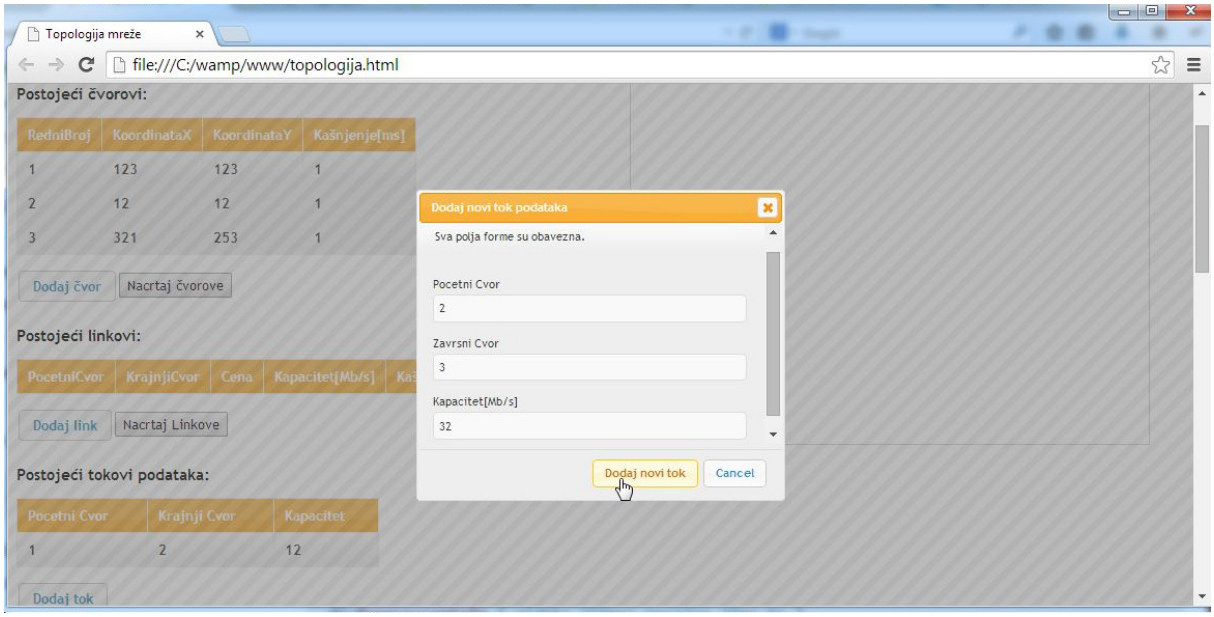
```
  var brojac=Cvorovi.length+1;
  $( "#dialog-node" ).dialog({
    autoOpen: false,
    height: 300,
    width: 350,
    modal: true,
    buttons: {
      "Dodaj novi čvor": function() {
         var NodeValid = true;
         NodeValid =
ValidateNodeParameters(c,KoordinataX.val(),KoordinataY.val(),KasnjenjeCvora.val(
));
         allFieldsNode.removeClass( "ui-state-error" );
         if ( NodeValid ) {
           $( "#nodes tbody" ).append( "<tr>" +
           "<td>" + brojac + "</td>" +
           "<td>" + KoordinataX.val() + "</td>" +
           "<td>" + KoordinataY.val() + "</td>" +
           "<td>" + KasnjenjeCvora.val() + "</td>" +
           "</tr>" );
           brojac=brojac+1;
           var NoviCvor= new
Cvor(KoordinataX.val(),KoordinataY.val(),KasnjenjeCvora.val());
           Cvorovi.push(NoviCvor);
           $( this ).dialog( "close" );
         }
        },
      Cancel: function() {
        $( this ).dialog( "close" );
       }
      },
      close: function() {
        allFieldsNode.val( "" ).removeClass( "ui-state-error" );
      }
  });
```

Kada se popune polja za parametre čvora u dijalogu i klikne na dugme "Dodaj novi čvor", proverava se validnost unetih podataka pomoću funkcije ***ValidateNodeParameters()***, opisane u četvrtom poglavlju, i ukoliko su parametri u ispravnom formatu, parametri čvora se dodaju u tabelu, kreira se novi objekat čvora pomoću funkcije ***Cvor()***, dodaje se u niz *Cvorovi* i dijalog se automatski zatvara. Ukoliko parametri nisu validni, korisnik se obaveštava porukom kako bi ispravio grešku, pri čemu se prozor ne zatvara automatski. Prozor može da se zatvori manuelno klikom na dugme "*Cancel*", ili na dugme "*close*", u gornjem desnom uglu. Promenljiva *brojac* koristi se za prikazivanje rednog broja unetog čvora.

Isti princip kao kod unošenja parametara čvora primenjen je i za link i tok. Navedeni kod je isti, samo su nazivi promenljivih drugačiji. Primer dodavanja novog linka može se videti na slici 5.1.4, a primer unošenja novog toka na stranici 5.1.5. Unošenje linka i toka zahteva da su prethodno uneti čvorovi koje oni povezuju.



Slika 5.1.4. Dodavanje novog linka



Slika 5.1.5. Dodavanje novog toka

Nakon unošenja čvorova i linkova, moguće je nacrtati čvorove i linkove u elementu *canvas*. Crtanje čvorova se aktivira klikom na dugme "Nacrtaj čvorove", koji se nalazi u HTML fajlu.

```
<button id="draw-node">Nacrtaj čvorove</button>
```

Tada se poziva funkcija ***CrtajCvorove()***:

```
$("#draw-node").click(function(){
   CrtajCvorove(c,ctx,Cvorovi);
 });
```

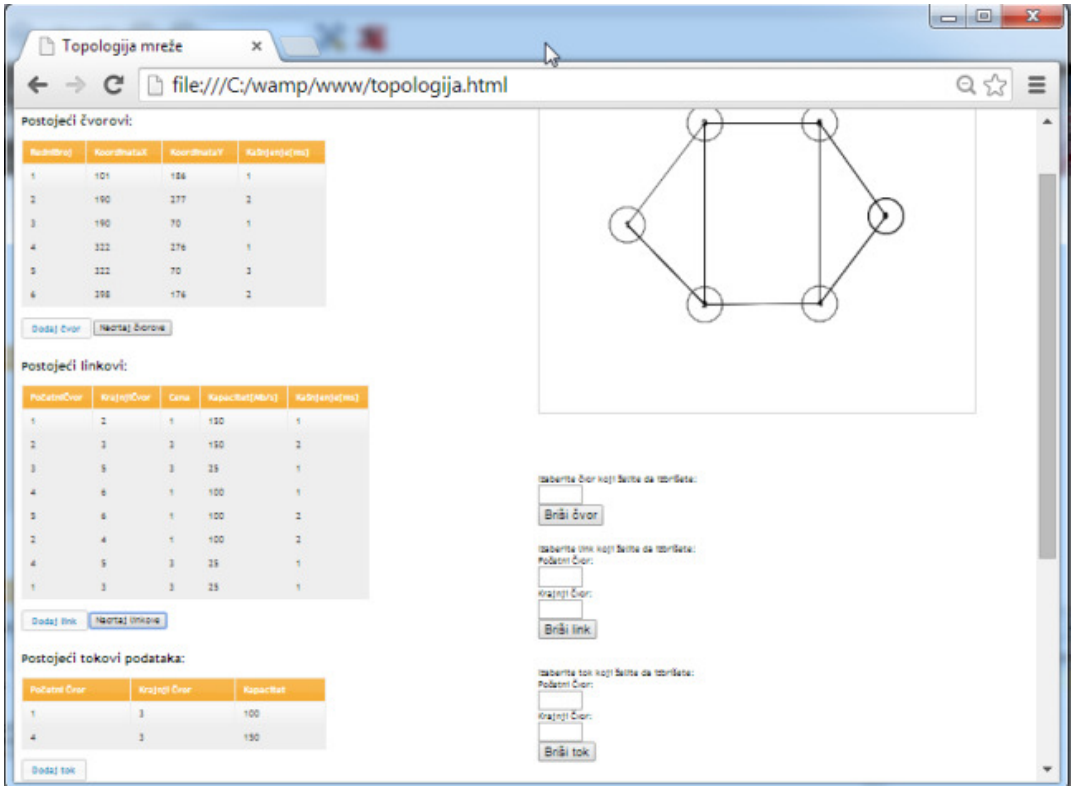
Ekvivalento tome, crtanje linkova se vrši klikom na dugme "Nacrtaj linkove":

```
<button id="draw-link">Nacrtaj Linkove</button>
```

Time se poziva funkcija ***CrtajLinkove()***:

```
$("#draw-link").click(function(){
   CrtajLinkove(c,ctx,Linkovi);
});
```

Primer crtanja topologije mreže prikazan je na slici 5.1.6.



Slika 5.1.6. Izgled topologije mreže

5.2. Brisanje čvora, linka i toka

5.2.1. Brisanje linka

Primer brisanja linka biće prikazan na topologiji sa slike 5.1.6. Da bi korisnik obrisao link, potrebno je da unese brojeve početnog i krajnjeg čvora linka kojeg želi da obriše. Polja za unos brojeva čvorova i dugme "Briši link" generisani su pomoću sledećeg HTML koda:

```
<form style="position:absolute; left:600px; top: 580px; right:30px;">
Izaberite link koji želite da izbrišete:</br>
```

```
Početni Čvor:<input type="number" min="1" id="nodeA">
Krajnji Čvor:<input type="number" min="1" id="nodeB">
<button type="button" id="brisiLink">Briši link</button>
</form>
```

Klikom na dugme “Briši link“ poziva se sledeća funkcija:

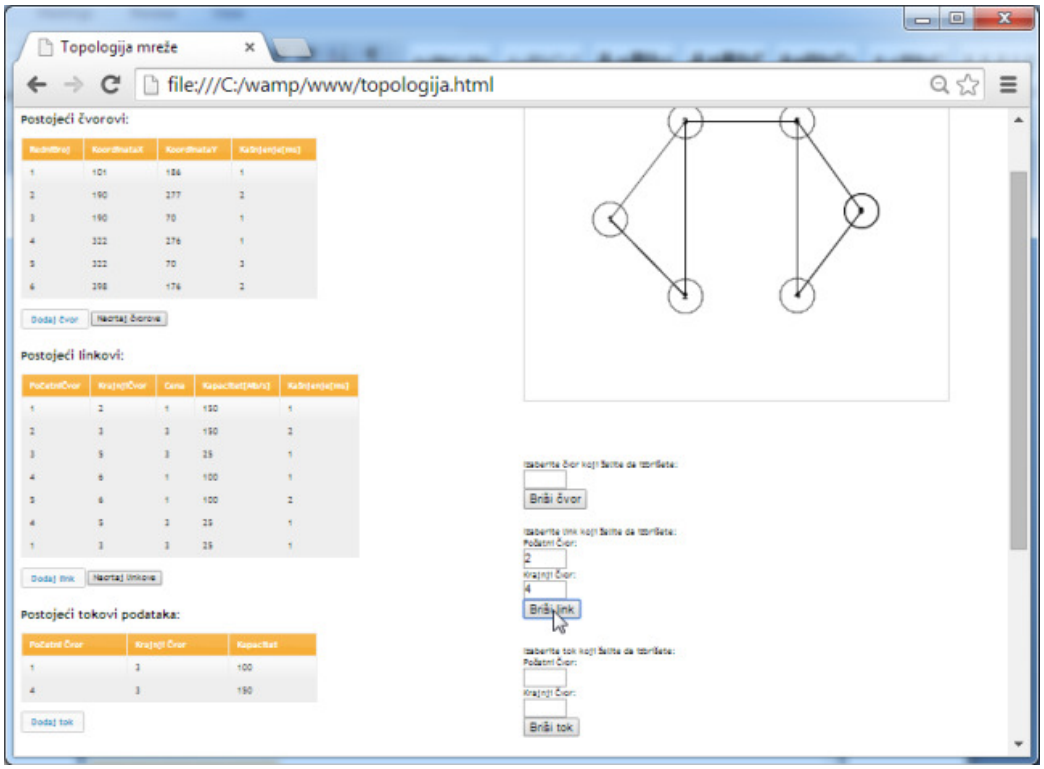
```
    $("#brisiLink").click(function()  {
      var NodeA=document.getElementById("nodeA").value;
      var NodeB=document.getElementById("nodeB").value;
      NodeA=parseInt(NodeA);
      NodeB=parseInt(NodeB);
      BrisiLink(NodeA,NodeB,c,ctx);
      CrtajSve(Cvorovi,Linkovi,Tokovi);
    })
```

Na početku funkcije se deklarišu promenljive *NodeA* i *NodeB*, kao vrednosti tekstualnih polja *nodeA* i *nodeB*, respektivno. Vrši se konverzija njihovih vrednosti u tip *integer*, i one se, zajedno sa referencom na element *canvas* i njegovim kontekstom, prosleđuju funkciji ***BrisiLink()***. Na kraju se pomoću funkcije ***CrtajSve()*** vrši ponovno ispisivanje parametara čvorova, linkova i tokova i iscrtavanje topologije na *canvas*-u. Kod funkcije ***CrtajSve()*** je sledeći:

```
function CrtajSve(Cvorovi,Linkovi,Tokovi)  {
  c=document.getElementById("MojCanvas");
  ctx=c.getContext("2d");
  CrtajCvorove(c,ctx,Cvorovi);
  CrtajLinkove(c,ctx,Linkovi);
  $("#nodes > tbody > tr").remove();
  for (i = 0; i < Cvorovi.length; i++) {
    Cvori=Cvorovi[i]; m=i+1;
    $( "#nodes tbody" ).append( "<tr>" +
      "<td>" + m + "</td>" +
      "<td>" + Cvori.KoordinataX + "</td>" +
      "<td>" + Cvori.KoordinataY + "</td>" +
      "<td>" + Cvori.KasnjenjeCvora + "</td>" +
      "</tr>" );
  }
  $("#links > tbody > tr").remove();
  for (i = 0; i < Linkovi.length; i++) {
    Linki=Linkovi[i];
    $( "#links tbody" ).append( "<tr>" +
      "<td>" + Linki.PocetniCvor + "</td>" +
      "<td>" + Linki.KrajnjiCvor + "</td>" +
      "<td>" + Linki.Cena + "</td>" +
      "<td>" + Linki.Kapacitet + "</td>" +
      "<td>" + Linki.KasnjenjeLinka + "</td>" +
      "</tr>" );
  }
  $("#flows > tbody > tr").remove();
  for (i = 0; i < Tokovi.length; i++) {
    Toki=Tokovi[i];
    $( "#flows tbody" ).append( "<tr>" +
      "<td>" + Toki.PocetniCvorToka + "</td>" +
      "<td>" + Toki.ZavrsniCvorToka + "</td>" +
      "<td>" + Toki.KapacitetToka + "</td>" +
      "</tr>" );
  }
```

```
}
```

Na slici 5.2.1.1 ilustrovan je izgled stranice nakon brisanja linka između čvorova 2 i 4, topologije prikazane na slici 5.1.6.



Slika 5.2.1.1. Brisanje linka između čvorova 2 i 4

5.2.2. Brisanje čvora

Brisanjem čvora brišu se linkovi i tokovi, koji povezuju dati čvor sa drugim čvorovima. Čvor se briše unosom rednog broja čvora koji treba da se obriše i klikom na dugme "Briši čvor", čiji je HTML kod sledeći:

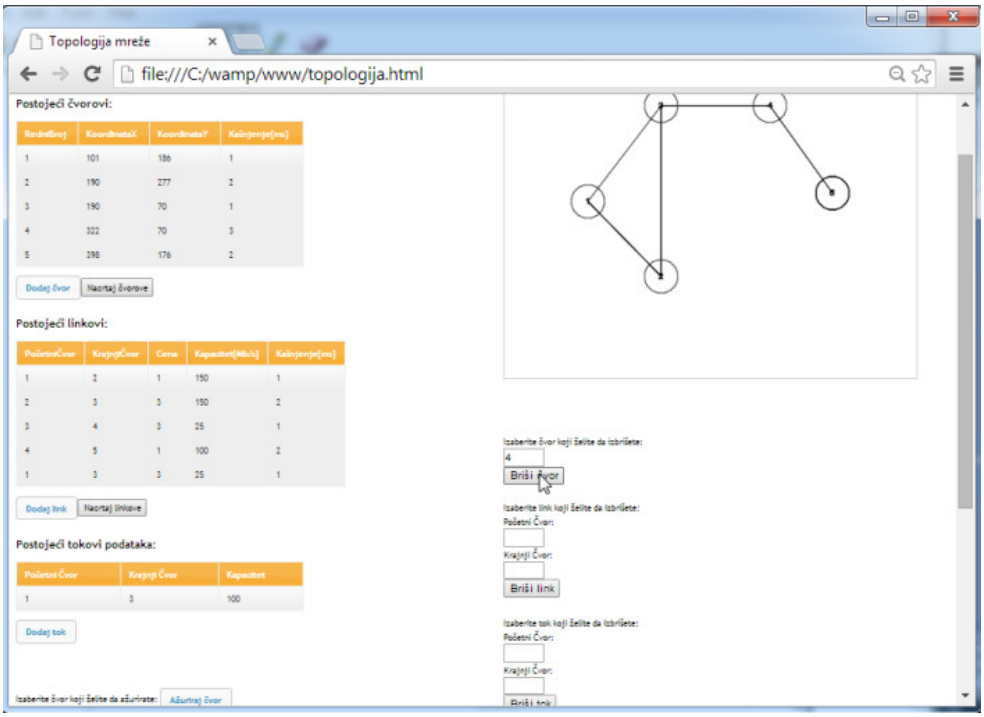
```
<form style="position:absolute; left:600px; top: 500px; right:30px;">
Izaberite čvor koji želite da izbrišete:
<input type="number" min="1" id="deleteNode">
<button type="button" id="brisiNode">Briši čvor</button>
</form>
```

Klikom na dugme "Briši čvor" poziva se funkcija čiji je kod prikazan ispod. U ovoj funkciji se deklariše promenljiva *num*, koja predstavlja redni broj čvora koji je korisnik uneo. Ona se, pored reference na *canvas* i konteksta *canvas*-a prosleđuje funkciji ***BrisiCvor()***, koja će obrisati dati čvor i odgovarajuće linkove i tokove iz nizova *Cvorovi*, *Linkovi* i *Tokovi*. Vrednost promenljive *brojac*, pomoću koje se ispisuje redni broj novog čvora u tabeli čvorova, se smanjuje za 1. Na kraju se poziva funkcija ***CrtajSve()***, ispisuje se sadržaj tabela postojećih čvorova, linkova i tokova i crta se mrežna topologija na *canvas*-u.

```
    $("#brisiNode").click(function() {
      var num=document.getElementById("deleteNode").value;
      num=parseInt(num);
      BrisiCvor(num,c,ctx);
```

```
        brojac=brojac-1;
        CrtajSve(Cvorovi,Linkovi,Tokovi);
    });
```

Na slici 5.2.2.1 prikazan je primer brisanja čvora sa rednim brojem 4, iz topologije prikazane na slici 5.2.1.1. Tom prilikom su obrisani i linkovi između čvorova 2 i 5 i čvorova i 4 i 6, kao i tok između čvorova 3 i 4, prikazani na slici 5.2.1.1.



Slika 5.2.2.1. Brisanje čvora pod rednim brojem 4

5.2.3. Brisanje toka

Da bi se obrisao tok, potrebno je uneti početni i krajnji čvor toka podataka u polja "Početni Čvor" i "Krajnji Čvor" i kliknuti na dugme "Briši tok", čiji je HTML kod sledeći:

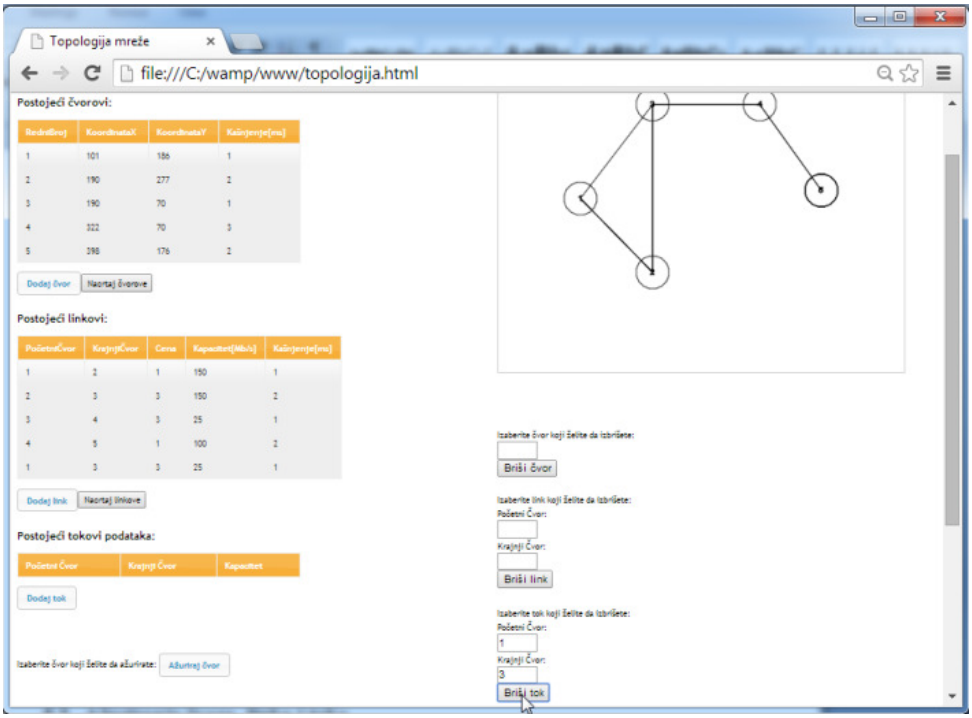
```
<form>
Izaberite tok koji želite da izbrišete:</br>
Početni Čvor: <input type="number" min="1" id="FlowNodeA">
Krajnji Čvor:<input type="number" min="1" id="FlowNodeB">
<button type="button" id="brisiTok">Briši tok</button>
</form>
```

Klikom na dugme "Briši tok" poziva se funkcija koja poziva funkciju ***BrisiTok()***. Ova funkcija briše željeni tok iz niza *Tokovi*, a pomoću funkcije ***CrtajSve()*** vrši se prikaz svih čvorova, linkova i tokova po tabelama, bez obrisanog toka, i iscrtavanje mrežne topologije na *canvas*-u.

```
    $("#brisiTok").click(function(){
        var x=document.getElementById("FlowNodeA").value;
        var y=document.getElementById("FlowNodeB").value;
        x=parseInt(x);
        y=parseInt(y);
        BrisiTok(x,y);
```

```
        CrtajSve(Cvorovi,Linkovi,Tokovi);
    })
```

Na slici 5.2.3.1 prikazan je izgled stranice odmah pošto su popunjena polja “Početni Čvor“ i “Krajnji Čvor“ i pošto je kliknuto na dugme “Briši tok“. Tok između čvorova 1 i 3 je obrisan iz niza *Tokovi* i uklonjen iz tabele postojećih tokova, što se može videti na slici:



Slika 5.2.3.1. Klikom na dugme Briši tok, obrisan je tok

5.3. Ažuriranje čvora, linka i toka

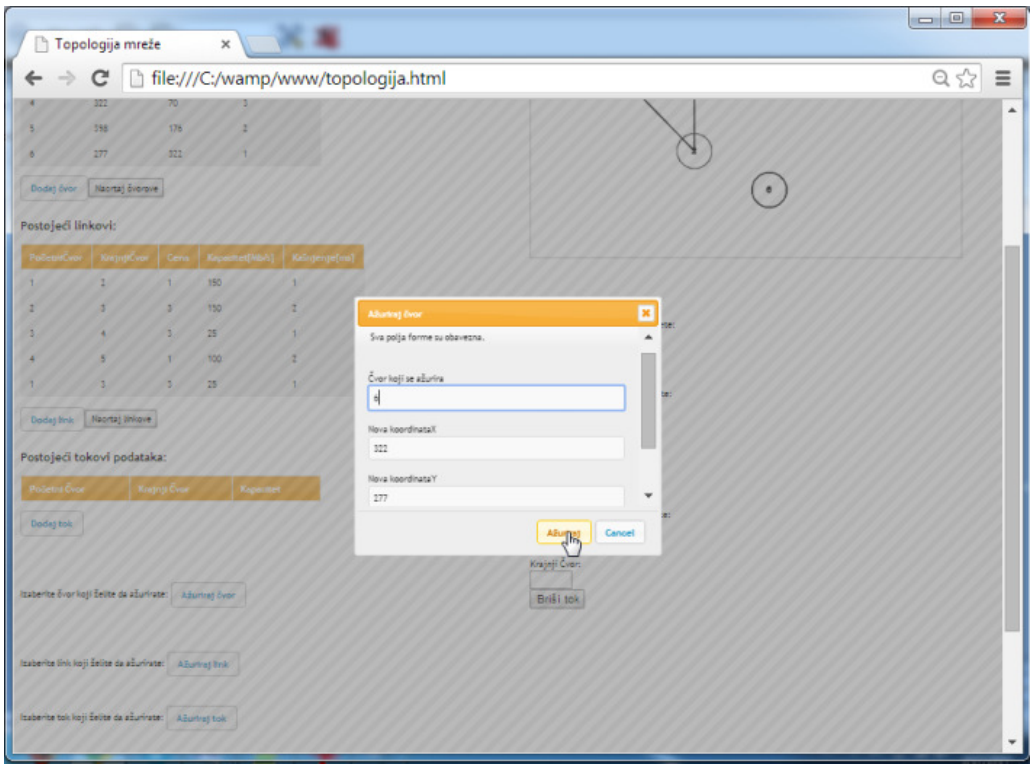
5.3.1. Ažuriranje čvora

Da bi se ažurirao čvor, treba kliknuti na dugme “Ažuriraj čvor“:

```
<form>
Izaberite čvor koji želite da ažurirate:
<button type="button" id="AzurirajCvor">Ažuriraj čvor</button></br></br>
</form>
```

Klikom na ovo dugme se poziva funkcija, čiji je kod prikazan ispod, i koja otvara otvara dijalog za ažuriranje čvora, slično kao pri dodavanju novog čvora. Dijalog je prikazan na slici 5.3.1. U njemu je neophodno popuniti redni broj čvora koji se ažurira i njegove nove parametre. Potrebno je uneti sve parametre, iako ne moraju biti različiti od već unetih.

```
$( "#AzurirajCvor" ).button().click(function() {
  $( "#dialog-node-update" ).dialog( "open" );
})
```



Slika 5.3.1.1. Ažuriranje čvora pod rednim brojem 6

HTML kod dijaloga je sledeći:

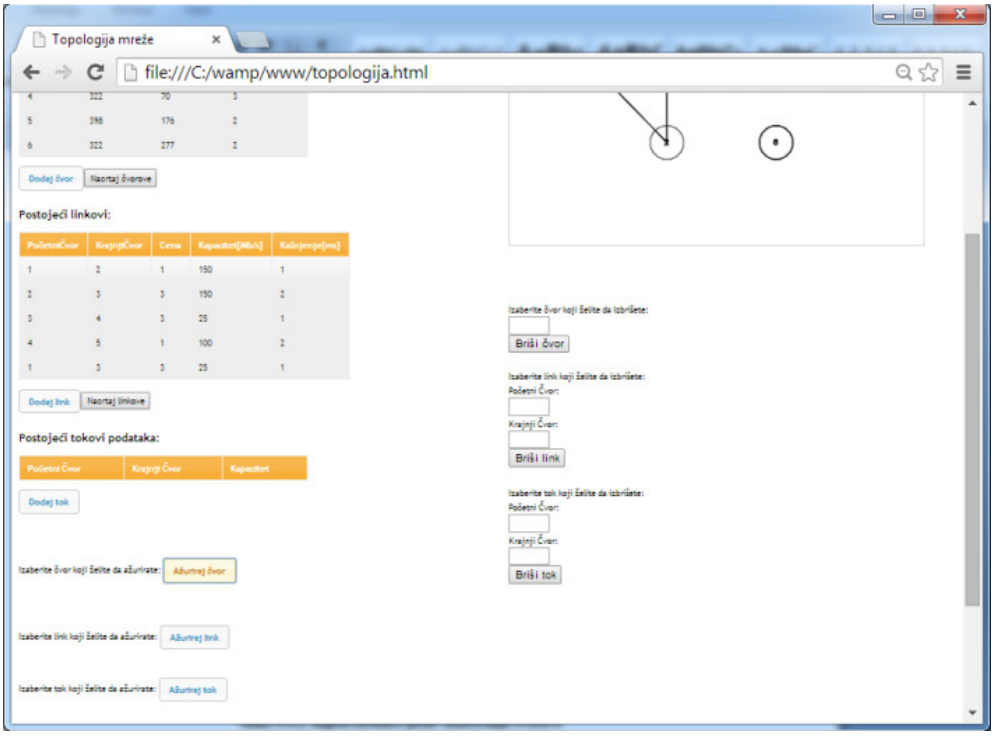
```
    <div id="dialog-node-update" title="Ažuriraj čvor">
        <p class="validateTips">Sva polja forme su obavezna.</p>
        <form>
            <fieldset>
                <label for="updateNode">Čvor koji se ažurira</label>
                <input type="number" min="1" name="updateNode" id="updateNode"
class="text ui-widget-content ui-corner-all" >
                <label for="X">Nova koordinataX</label>
                <input type="number" min="0" name="X" id="X" class="text ui-
widget-content ui-corner-all" >
                <label for="Y">Nova koordinataY</label>
                <input type="number" min="0" name="Y" id="Y" value=""
class="text ui-widget-content ui-corner-all" >
                <label for="AzuriranjeKasnjenja">Novo kašnjenje[ms]</label>
                <input type="number" name="AzuriranjeKasnjenja"
id="AzuriranjeKasnjenja" value="" class="text ui-widget-content ui-corner-all" >
            </fieldset>
        </form>
    </div>
```

Dijalog i ažuriranje čvora su definisani funkcijom čiji je kod prikazan ispod. Klikom na dugme "Ažuriraj" u dijalogu, pomoću funkcije ***ValidateUpdateNodeParameters()***, opisane u odeljku 4.5.1, proverava se ispravnost unetih podataka i ukoliko su podaci ispravni i funkcija vrati vrednost "*true*", kreiraće se objekat *NoviCvor*, sa parametrima koje je korisnik uneo u dijalogu, i ažuriraće se niz *Cvorovi*, pomoću funkcije ***AzurirajNode().*** Na kraju se vrši iscrtavanje čvorova i linkova na *canvas*-u i prikazivanje tabele postojećih čvorova, kako bi se uvidele unete izmene. Ukoliko podaci

koje je korisnik uneo nisu ispravni, korisniku će se ispisati greška, a dijalog će ostati otvoren kako bi se greška ispravila. Klikom na "*Cancel*" ili "*close*" u gornjem desnom uglu, dijalog se zatvara.

```
  $( "#dialog-node-update" ).dialog({
    autoOpen: false,
    height: 300,
    width: 350,
    modal: true,
    buttons: {
    "Ažuriraj": function() {
      var UpdateNodeValid = true;
      var updateNode = $( "#updateNode" ),
      X = $( "#X" ),
      Y = $( "#Y" ),
      AzuriranjeKasnjenja = $( "#AzuriranjeKasnjenja" ),
      allFieldsNode = $( [] ).add( updateNode ).add( X ).add( Y ).add(
AzuriranjeKasnjenja );
      allFieldsNode.removeClass( "ui-state-error" );
      UpdateNodeValid =
ValidateUpdateNodeParameters(updateNode.val(),c,X.val(),Y.val(),AzuriranjeKasnje
nja.val());
      if ( UpdateNodeValid ) {
        updateNode=updateNode.val();
        updateNode=parseInt(updateNode);
        var NoviCvor= new Cvor(X.val(),Y.val(),AzuriranjeKasnjenja.val());
        AzurirajNode(updateNode,NoviCvor);
        CrtajCvorove(c,ctx,Cvorovi);
        CrtajLinkove(c,ctx,Linkovi);
        $("#nodes > tbody > tr").remove();
        for (i = 0; i < Cvorovi.length; i++) {
          Cvori=Cvorovi[i]; m=i+1;
          $( "#nodes tbody" ).append( "<tr>" +
            "<td>" + m + "</td>" +
            "<td>" + Cvori.KoordinataX + "</td>" +
            "<td>" + Cvori.KoordinataY + "</td>" +
            "<td>" + Cvori.KasnjenjeCvora + "</td>" +
            "</tr>" );
        }
      $( this ).dialog( "close" );
      }
      },
      Cancel: function() {
           $( this ).dialog( "close" );
      }
      },
      close: function() {
           allFieldsNode.val( "" ).removeClass( "ui-state-error" );
      }
    });
```

Na slici 5.3.1.2 se mogu videti izmene nakon ažuriranja koordinata čvora pod rednim brojem 6. Nove koordinate čvora, 322 i 277 su upisane u tabelu, a na *canvas*-u je čvor 6 zauzeo novu poziciju.



Slika 5.3.1.2. Izgled stranice posle ažuriranja čvora 6

5.3.2. Ažuriranje linka

Za ažuriranje linka korisnik treba da klikne na dugme “Ažuriraj link“:

```
<form>
Izaberite link koji želite da ažurirate:
<button type="button" id="AzurirajLink">Ažuriraj link</button>
</form>
```

Tada se poziva sledeća funkcija:

```
$( "#AzurirajLink" ).button().click(function() {
  $( "#dialog-link-update" ).dialog( "open" );
})
```

Slično kao kod ažuriranja čvora, otvara se dijalog, čiji je HTML kod sledeći:

```
    <div id="dialog-link-update" title="Ažuriraj Link">
        <p class="validateTips">Sva polja forme su obavezna.</p>
        <form>
            <fieldset>
                <label for="updateLinkNodeA">Početni čvor linka koji se
ažurira</label>
                <input type="number" min="1" name="updateLinkNodeA"
id="updateLinkNodeA" class="text ui-widget-content ui-corner-all" >
                <label for="updateLinkNodeB">Krajnji čvor linka koji se
ažurira</label>
                <input type="number" min="1" name="updateLinkNodeB"
id="updateLinkNodeB" class="text ui-widget-content ui-corner-all" >
                <label for="NovaCena">Nova cena</label>
                <input type="number" name="NovaCena" id="NovaCena" value=""
class="text ui-widget-content ui-corner-all" >
```

```
                <label for="NoviKapacitet">Novi kapacitet[Mb/s]</label>
                <input type="number" name="NoviKapacitet" id="NoviKapacitet"
value="" class="text ui-widget-content ui-corner-all" >
                <label for="NovoKasnjenjeLinka">Novo kašnjenje[ms]</label>
                <input type="number" name="NovoKasnjenjeLinka"
id="NovoKasnjenjeLinka" value="" class="text ui-widget-content ui-corner-all" >
            </fieldset>
        </form>
    </div>
```

U dijalog, koji se otvara klikom na dugme "Ažuriraj link" i koji se može videti na slici 5.3.2.1, treba uneti početni i krajnji čvor linka koji se ažurira i novi kapacitet, novo kašnjanje ili novu cenu. Sledeća funkcija definiše ponašanje dijaloga:

```
  $( "#dialog-link-update" ).dialog({
    autoOpen: false,
    height: 300,
    width: 350,
    modal: true,
    buttons: {
      "Ažuriraj": function() {
         var UpdateLinkValid = true;
         var updateLinkNodeA = $( "#updateLinkNodeA" ),
         updateLinkNodeB = $( "#updateLinkNodeB" ),
         NovaCena = $( "#NovaCena" ),
         NoviKapacitet = $( "#NoviKapacitet" ),
         NovoKasnjenjeLinka = $( "#NovoKasnjenjeLinka" ),
         allFieldsLink = $( [] ).add( updateLinkNodeA ).add( updateLinkNodeB
).add( NovaCena ).add( NoviKapacitet ).add( NovoKasnjenjeLinka );
         allFieldsLink.removeClass( "ui-state-error" );
         UpdateLinkValid =
ValidateUpdateLinkParameters(updateLinkNodeA.val(),updateLinkNodeB.val(),
NovaCena.val(),NoviKapacitet.val(),NovoKasnjenjeLinka.val());
         if ( UpdateLinkValid ) {
           var NoviLink= new
Link(updateLinkNodeA.val(),updateLinkNodeB.val(),NovaCena.val(),NoviKapacitet.va
l(),NovoKasnjenjeLinka.val());
           updateLinkNodeA=updateLinkNodeA.val();
           updateLinkNodeA=parseInt(updateLinkNodeA);
           updateLinkNodeB=updateLinkNodeB.val();
           updateLinkNodeB=parseInt(updateLinkNodeB);
           AzurirajLink(updateLinkNodeA,updateLinkNodeB,NoviLink);
           $("#links > tbody > tr").remove();
           for (i = 0; i < Linkovi.length; i++) {
             Linki=Linkovi[i];
             $( "#links tbody" ).append( "<tr>" +
               "<td>" + Linki.PocetniCvor + "</td>" +
               "<td>" + Linki.KrajnjiCvor + "</td>" +
               "<td>" + Linki.Cena + "</td>" +
               "<td>" + Linki.Kapacitet + "</td>" +
               "<td>" + Linki.KasnjenjeLinka + "</td>" +
               "</tr>" );
           }

         $( this ).dialog( "close" );
         }
        },
         Cancel: function() {
```

```
            $( this ).dialog( "close" );
          }
        },
    close: function() {
            allFieldsLink.val( "" ).removeClass( "ui-state-error" );
    }
    });
```

Klikom na dugme "Ažuriraj" poziva se funkcija u kojoj se deklariše promenljiva *UpdateLinkValid*, kojoj se dodeljuje vrednost koju vraća funkcija ***ValidateUpdateLinkParameters()***, opisana u odeljku 4.5.2. Ova funkcija proverava ispravnost podataka unetih u dijalogu, a ti podaci su početni i krajnji čvor linka koji se ažurira, kašnjenje, kapacitet i cena. Ako *UpdateLinkValid* ima vrednost "*true*", odnosno ako su podaci u ispravnom formatu, onda se kreira objekat *NoviLink* i pomoću funkcije ***AzurirajLink()*** ažurira se niz *Linkovi*. Na kraju se ponovo ispisuje tabela postojećih linkova, kako bi se prikazale izmene. Klikom na "*Cancel*" i na "*close*" u gornjem desnom uglu, dijalog se zatvara.

Na slici 5.3.2.1 je prikazan dijalog za ažuriranje linka. Ažurira se link između čvorova 2 i 6, sa cenom 1, kapacitetom 100 i kašnjenjem 3. Za novu cenu postavljena je vrednost 3, za kapacitet 150, a za kašnjenje 2, a zatim je kliknuto na dugme "Ažuriraj". Izgled tabele postojećih linkova i nove vrednosti parametara linka između čvorova 2 i 6, nakon ažuriranja linka, mogu se videti na slici 5.3.2.2.

5.3.3. Ažuriranje toka

Tok se ažurira klikom na dugme "Ažuriraj tok", kada se otvara dijalog prikazan na slici 5.3.3.1, u koji je potrebno uneti redne brojeve čvorova koji se ažuriraju i novi kapacitet toka. HTML kod kojim je prikazano dugme "Ažuriraj tok" je sledeći:

```
<form>
Izaberite tok koji želite da ažurirate:
<button type="button" id="AzurirajTok">Ažuriraj tok</button>
</form>
```

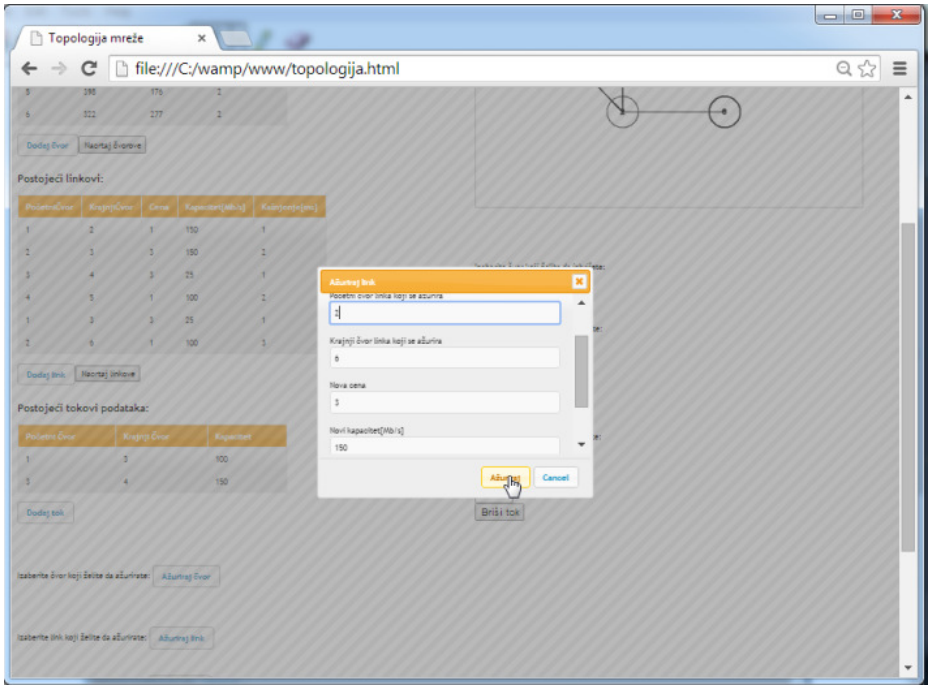
Funkcija koja se poziva klikom na dugme "Ažuriraj tok" i koja otvara dijalog je:

```
$( "#AzurirajTok" ).button().click(function() {
  $( "#dialog-flow-update" ).dialog( "open" );
})
```

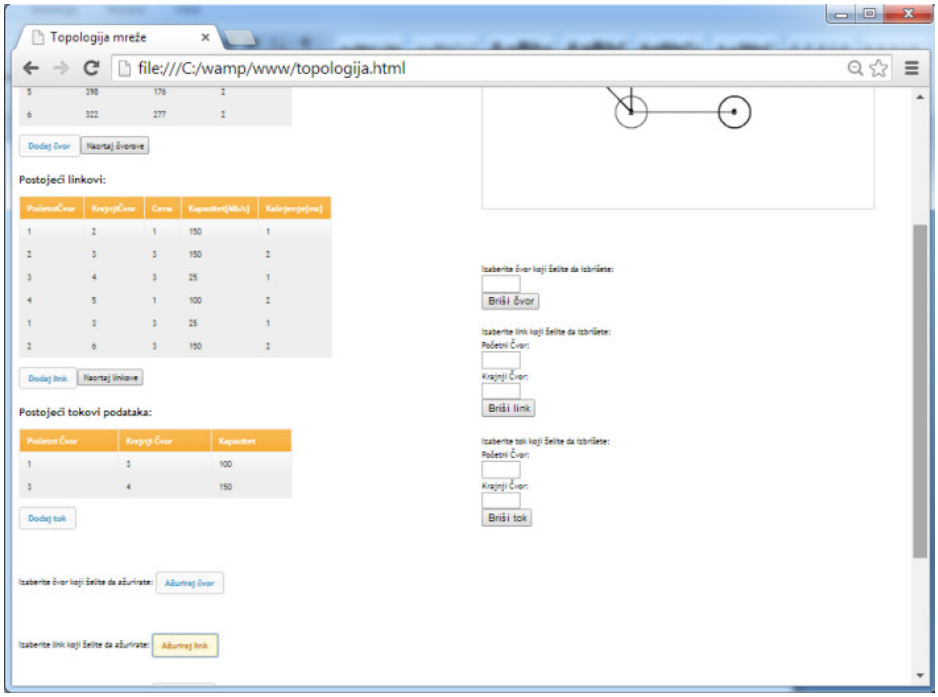
HTML kod dijaloga za ažuriranje toka je:

```
    <div id="dialog-flow-update" title="Ažuriraj Tok">
        <p class="validateTips">Sva polja forme su obavezna.</p>
        <form>
            <fieldset>
                <label for="updateFlowNodeA">Početni čvor toka koji se
ažurira</label>
                <input type="number" min="1" name="updateFlowNodeA"
id="updateFlowNodeA" class="text ui-widget-content ui-corner-all" >
                <label for="updateFlowNodeB">Krajnji čvor linka koji se
ažurira</label>
                <input type="number" min="1" name="updateFlowNodeB"
id="updateFlowNodeB" class="text ui-widget-content ui-corner-all" >
                <label for="newFlowCapacity">Novi kapacitet[Mb/s]</label>
```

```
                <input type="number" name="newFlowCapacity" id="newFlowCapacity"
value="" class="text ui-widget-content ui-corner-all">
            </fieldset>
        </form>
    </div>
```



Slika 5.3.2.1. Ažuriranje linka između čvorova 2 i 6

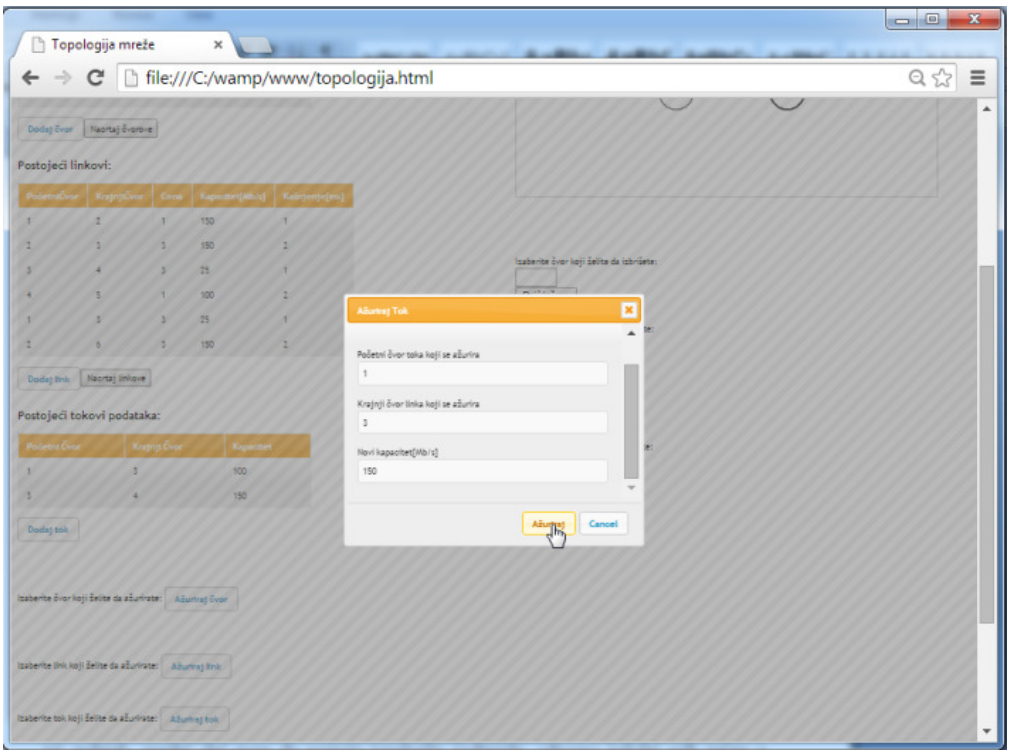


5.3.2.2 Izgled stranice posle ažuriranja linka između čvorova 2 i 6

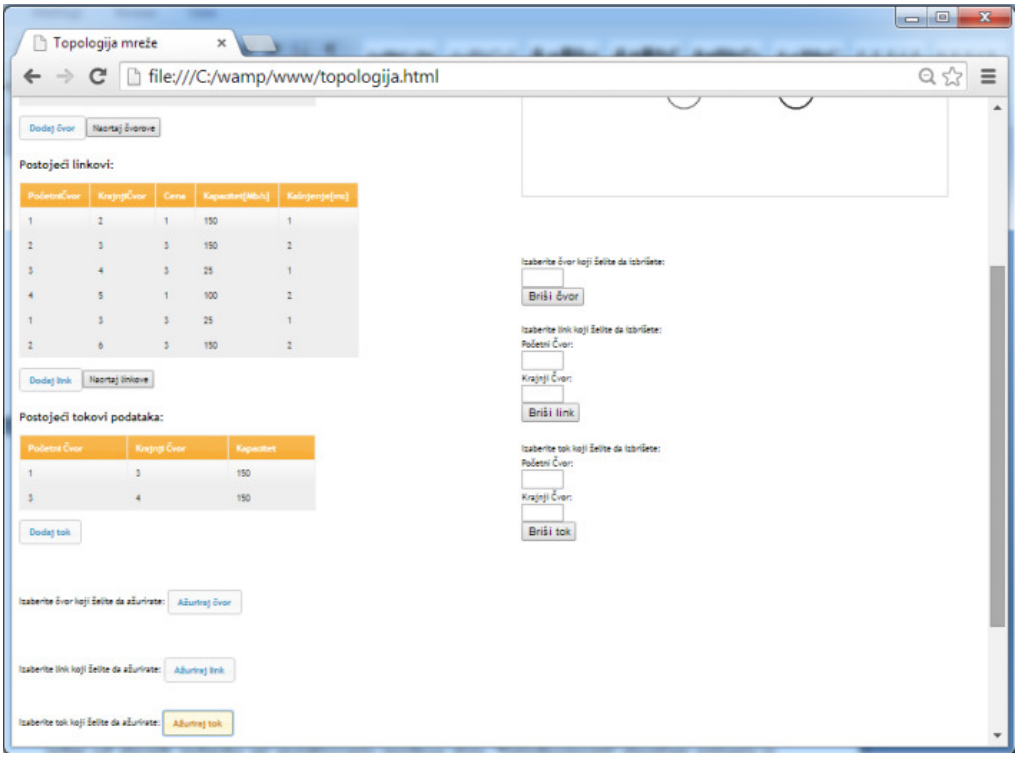
Ponašanje dijaloga je definisano funkcijom čiji je kod prikazan ispod. Kada se klikne na dugme Ažuriraj u dijalogu, proverava se ispravnost prosleđenih podataka unetih u dijalogu pomoću funkcije ***ValidateUpdateFlowParameters()*** i ukoliko su parametri ispravni pristupa se ažuriranju niza *Tokovi* pomoću funkcije ***AžurirajTok()***. Nakon toga sledi kod za ponovno ispisivanje tabele postojećih tokova kako bi se prikazale izmene.

```
  $( "#dialog-flow-update" ).dialog({
    autoOpen: false,
    height: 300,
    width: 350,
    modal: true,
    buttons: {
    "Ažuriraj": function() {
      var UpdateFlowValid = true;
      var updateFlowNodeA = $( "#updateFlowNodeA" ),
      updateFlowNodeB = $( "#updateFlowNodeB" ),
      newFlowCapacity = $( "#newFlowCapacity" ),
      allFieldsFlow = $( [] ).add( updateFlowNodeA ).add( updateFlowNodeB ).add(
newFlowCapacity );
      allFieldsFlow.removeClass( "ui-state-error" );
      UpdateFlowValid =
ValidateUpdateFlowParameters(updateFlowNodeA.val(),updateFlowNodeB.val(),
newFlowCapacity.val());
      if ( UpdateFlowValid ) {
        var NoviTok= new
Tok(updateFlowNodeA.val(),updateFlowNodeB.val(),newFlowCapacity.val());
        updateFlowNodeA=updateFlowNodeA.val();
        updateFlowNodeA=parseInt(updateFlowNodeA);
        updateFlowNodeB=updateFlowNodeB.val();
        updateFlowNodeB=parseInt(updateFlowNodeB);
        AzurirajTok(updateFlowNodeA,updateFlowNodeB,NoviTok);
        $("#flows > tbody > tr").remove();
        for (i = 0; i < Tokovi.length; i++) {
          Toki=Tokovi[i];
          $( "#flows tbody" ).append( "<tr>" +
            "<td>" + Toki.PocetniCvorToka + "</td>" +
            "<td>" + Toki.ZavrsniCvorToka + "</td>" +
            "<td>" + Toki.KapacitetToka + "</td>" +
            "</tr>" );
        }

         $( this ).dialog( "close" );
        }
        },
         Cancel: function() {
          $( this ).dialog( "close" );
        }
        },
    close: function() {
          allFieldsFlow.val( "" ).removeClass( "ui-state-error" );
    }
    });
```

Na slici 5.3.3.1 u tabeli postojećih tokova se može videti da kapacitet toka između čvorova 1 i 3 ima vrednost 100 i da je u dijalogu upisana vrednost 150 kao novi kapacitet između čvorova 1 i 3. Posle klika na dugme “Ažuriraj“ u dijalogu, prikazana je nova vrednost kapaciteta ovog toka, a to je 150, što se može videti na slici 5.3.3.2.



Slika 5.3.3.1. Ažuriranje toka između čvorova 1 i 3



Slika 5.3.3.2. Izgled stranice nakon ažuriranja toka

5.4. Primena Dijkstra algoritma na selektovani čvor

Funkcija mrežnog sloja OSI referentnog modela je da pakete sprovede od izvora do odredišta, između kojih može da se nalazi više različitih putanja kroz veći broj rutera. Da bi ispunio taj zadatak, mrežni sloj mora da poznaje topologiju podmreže, odnosno položaj svih rutera u podmreži, i da kroz mrežu bira odgovarajuće putanje, odnosno rute. On takođe mora da vodi računa da neke linkove i rutere ne preoptereti, a drugi da ne rade ništa. Algoritmi za rutiranje predstavljaju jedan od glavnih zadataka pri projektovanju mrežnog sloja. Najjednostavniji algoritam rutiranja je svakako Dijkstra algoritam, algoritam najkraće putanje, na kome su bazirani OSPF (Open *Shortest Path First*) i IS-IS (*Intermediate System to Intermediate System*) protokoli rutiranja.

Dijkstra algoritam funkcioniše tako što se napravi graf podmreže u kome svaki ruter predstavlja čvor, a svaki link jedan luk. Za dati čvor u grafu, pronalazi se putanja sa najmanjom cenom između tog čvora i svakog drugog čvora u mreži.

Funkcija pomoću koje je Dijkstra algoritam realizovan data je sledećim kodom:

```
    $("#dijkstra").click(function() {
      var RadniCvor=document.getElementById("pocetni").value;
      var CvoroviUStablu=[];
      CvoroviUStablu.push(RadniCvor);
      var LinkoviUStablu=[];
      var trencena=0;
      var cene=[];
      var preth=[];
      var susedi=[];
      while(CvoroviUStablu.length<Cvorovi.length) {
        k=0;
        while (k<Linkovi.length) {
          var Linki=Linkovi[k];
          Linki.Cena=parseInt(Linki.Cena);
          if((RadniCvor==Linki.PocetniCvor))  {
            var Q=Linki.KrajnjiCvor;
              if (CvoroviUStablu.indexOf(Q)<0)  {
                susedi.push(Q);
                cene.push(trencena+Linki.Cena);
                preth.push(RadniCvor);
              }
          }
          if((RadniCvor==Linki.KrajnjiCvor))  {
            var Q=Linki.PocetniCvor;
              if (CvoroviUStablu.indexOf(Q)<0)  {
                susedi.push(Q);
                cene.push(trencena+Linki.Cena);
                preth.push(RadniCvor);
              }
          }
        k++;
        }
        var min=cene[0];
        j=0;
        i=1;
        while (i<cene.length)  {
          if (min>cene[i]) {
             min=cene[i];
             j=i;
          }
```

```
        i++;
        }

          for(var i = 0; i < Linkovi.length ; i++) {
          var Linki=Linkovi[i];
          if (((Linki.PocetniCvor==susedi[j]) && (Linki.KrajnjiCvor==preth[j]))
|| ((Linki.PocetniCvor==preth[j]) && (Linki.KrajnjiCvor==susedi[j]))){
            LinkoviUStablu.push(Linki);
          }
        }

        CvoroviUStablu.push(susedi[j]);
        trencena=min;
        RadniCvor=susedi[j];

        var susedi_out=[];
        var l=0;
        while (l<susedi.length)  {
          if (susedi[j]==susedi[l]) {
            susedi_out.push(l);
          }
         l++;
        }

        for(var i = susedi.length - 1; i >= 0; i--) {
          if (susedi_out.indexOf(i)>=0) {
            susedi.splice(i, 1);
            cene.splice(i, 1);
            preth.splice(i, 1);
          }
        }
        }

      CrtajCvorove(c,ctx,Cvorovi);
      CrtajLinkove(c,ctx,LinkoviUStablu);
    })
```

Ova funkcija se poziva klikom na dugme "dijkstra" u HTML kodu i za njeno funkcionisanje neophodno je da korisnik unese u *<input>* polje inicijalni čvor, za koji se računaju nakraće putanje do svih ostalih čvorova u mreži:

```
<form>
Izaberite početni čvor za Dijkstra algoritam:
<input type="number" min="1" id="pocetni">
<button type="button" id="dijkstra">Dijkstra start</button>
</form>
```

Polje za unos inicijalnog čvora je označeno sa *id="pocetni"* i njegova vrednost se dodeljuje promenljivoj koja se naziva *RadniCvor*. Ova promenljiva će se tokom prolaska kroz WHILE petlje postajati poslednji čvor koji je dodat u stablo, odnosno poslednji čvor do kojeg je određena najbolja putanja od inicijalnog čvora. Neka se u daljem tekstu poslednji čvor do kojeg je određena najbolja putanja od inicjalnog čvora zove tekući radni čvor, a čvorovi do kojih je određena najbolja putanja pre tekućeg radnog čvora neka se zovu prethodni radni čvorovi. Zatim sledi inicijalizacija promenljivih *CvoroviUStablu*, *LinkoviUStablu*, *cene*, *preth*, *susedi* i *trencena*. *CvoroviUStablu* je definisan kao prazan niz u koji će se dodavati čvorovi topologije, tokom prolaska kroz petlju.

LinkoviUStablu je niz linkova koji će činiti najbolju putanju. Promenljiva *trencena* je cena od inicijalnog čvora do tekućeg radnog čvora. Niz *susedi* je niz susednih čvorova tekućeg radnog čvora i susednih čvorova prethodnih radnih čvorova. Niz *preth* je niz tekućih i prethodnih radnih čvorova, a *cene* niz cena putanja od inicijalnog čvora do susednog čvora tekućeg radnog čvora i od inicijalnog čvora do susednog čvora prethodnog radnog čvora. Elementi ova tri niza sa istim indeksom, određuju link između tekućeg (prethodnog) radnog čvora i njegovog suseda, kao i cenu putanje do susednog čvora tekućeg (prethodnog) radnog čvora.

Nakon inicijalizacije parametara definiše se spoljašnja petlja koja će se izvršavati sve dok dužina niza *CvoroviUStablu* ne postane jednaka dužini niza *Cvorovi*, odnosno dok svi čvorovi grafa mreže ne budu uključeni u stablo.

Unutar spoljašnje WHILE petlje definiše se unutrašnja WHILE petlja, koja prolazi kroz sve linkove niza *Linkovi* u potrazi sa susednim čvorovima tekućeg radnog čvora. Čvor je sused tekućem radnom čvoru ako postoji link koji ih povezuje. Zato se za svaki link proverava uslov da li je jedan njegov kraj *RadniCvor* i ako jeste, drugi čvor se proglašava za susedni, *Q*. Pošto su linkovi dvosmerni i *RadniCvor* može biti početni ili krajnji čvor jednog linka, proveravaju se oba ova slučaja, zbog čega postoje dve IF naredbe. Ako je ustanovljeno da je čvor susedni, proverava se još jedan uslov, a to je da li je *Q* već uključen u stablo, pomoću funkcije ***indexOf()***. Ova funkcija proverava da li je njen argument, u ovom slučaju *Q*, član niza *CvoroviUStablu*. Ukoliko *Q* nije pronađen u nizu *CvoroviUStablu*, funkcija ***indexOf()*** vraća -1, a u suprotnom indeks elementa *Q* u datom nizu. Ako je *Q* već uključen u stablo, nema svrhe razmatrati ovaj link, a ukoliko nije, stavlja se na kraj niza susedi pomoću funkcije ***push()***. Tekući *RadniCvor* se dodaje nizu *preth*, a cena putanje od inicijalnog čvora do susednog čvora tekućeg radnog čvora, koja je jednaka zbiru trenutne cene i cene tekućeg linka u iteraciji, se dodaje nizu *cene*, isto pomoću funkcije ***push()***. Ovim je završena unutrašnja WHILE petlja, koja je za tekući radni čvor odredila njegove susedne čvorove i cene putanja do tih susednih čvorova. Elementi dobijena tri niza, *preth*, *susedi* i *cene*, sa istim indeksom određuju grane sa cenama i između njih treba odrediti onu granu, koja uključena u stablo, daje putanju sa najmanjom cenom. To znači da treba naći granu sa najmanjom cenom, odnosno minimum niza *cene* i indeks tog minimuma radi identifikacije grane. Upravo to je urađeno u sledećem delu koda:

```
min=cene[0];
j=0;
i=1;
while (i<cene.length)  {
if (min>cene[i]) {
min=cene[i];
j=i;
}
i++;
}
```

Promenljivom *min* je označena najmanja cena iz niza *cene*, a promenljivom *j* indeks elementa *min*. Indeksom *j* određeni su početni i krajnji čvor linka koga treba uključiti u stablo, pa se u FOR ciklusu prolazi kroz sve članove niza *Linkovi*, ne bi li se odredio taj link i dodao u stablo, odnosno u niz *LinkoviUStablu*. U stablo se dodaje susedni čvor tekućeg (prethodnog) radnog čvora sa najmanjom cenom, *susedi[j]*, *RadniCvor* postaje *susedi[j]*, a cena putanje do novog radnog čvora je *min*:

```
CvoroviUStablu.push(susedi[j]);
trencena=min;
RadniCvor=susedi[j];
```

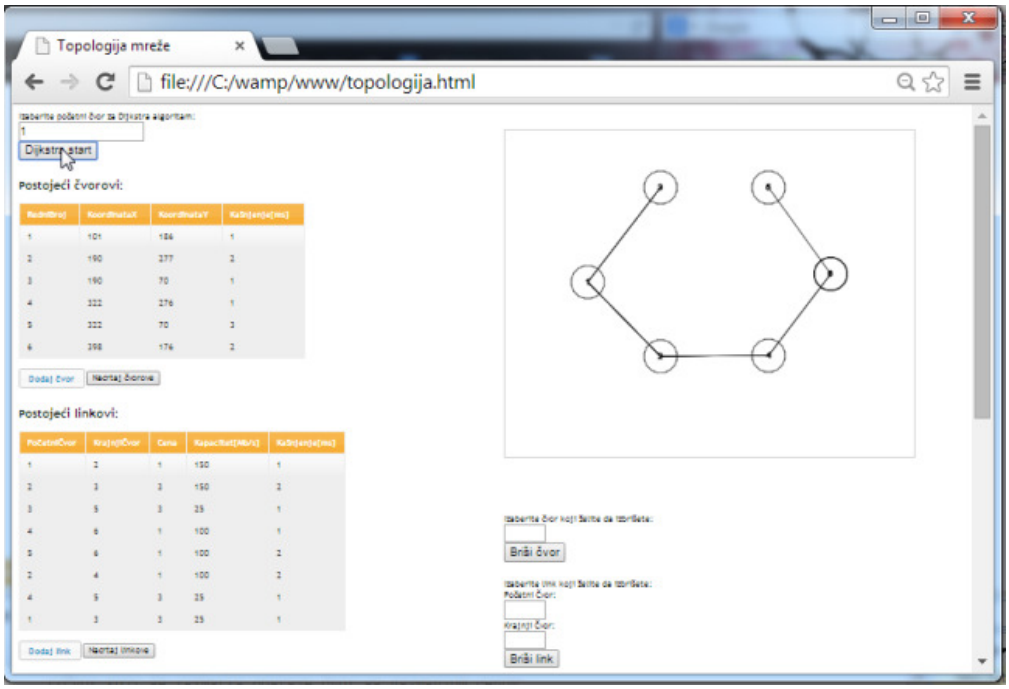
Kako u nizu susedi moze postojati više vrednosti *susedi[j]*, neophodno je isključiti iz razmatranja ove radne čvorove, da ne bi došlo do toga da ovaj čvor ponovo postane radni čvor, čime bi funkcija dala netačan rezultat. Za to se koristi niz *susedi_out*. Dakle, u ovaj niz su smešteni svi indeksi iz niza *susedi* koji odgovaraju elementima ovog niza koji su jednaki vrednosti *susedi[j]*. Elemente sa tim indeksima treba ukloniti iz nizova *susedi*, *preth* i *cene*, što je i urađeno u sledećem FOR ciklusu:

```
for(var i = susedi.length - 1; i >= 0; i--) {
  if (susedi_out.indexOf(i)>=0) {
     susedi.splice(i, 1);
     cene.splice(i, 1);
     preth.splice(i, 1);
     }
}
```

Prolazi se kroz sve elemente niza *susedi*, počev od poslednjeg elementa, i ukoliko se indeks trenutno posmatranog elementa nalazi u skupu indeksa koje treba ukloniti, taj element se uklanja pomoću funkcije ***splice()***, iz sva tri niza, *susedi*, *cene* i *preth*.

Time je završen spoljašnji WHILE ciklus i na kraju preostaje da se najbolja putanja od inicijalnog čvora do svakog čvora u datoj mreži i prikaže u elementu *canvas*, što se izvršava pozivanjem funkcija ***CrtajCvorove()*** i ***CrtajLinkove()***.

Primer Dijkstra algoritma na topologiju sa slike 5.1.6. primenjen na čvor pod rednim brojem 1, prikazan je na slici 5.4.1.



Slika 5.4.1. Dijkstra algoritam primenjen na čvor 1 mreže sa slike 5.6

5.5. Upisivanje topologije u bazu podataka

Za snimanje topologije korišćeni su *Apache* Veb server, MySQL baza podataka i PHP programski jezik.

Kreirana baza podataka ima naziv "*topology*" i četiri kolone. To su *Networks*, *Nodes*, *Links* i *Flows*. U tabeli *Networks* se čuvaju nazivi mreža, a u tabelama *Nodes*, *Links* i *Flows* spisak čvorova, linkova i tokova, respektivno, sa njihovim parametrima i identifikacijom mreže kojoj pripadaju.

Tabela *Networks* ima sledeće kolone:

- *NetworkID* – Jedinstveni ID mreže u vidu devetocifarnog broja. Prilikom dodavanja nove teme, vrednost ove kolone se automatski ažurira, jer je korišćena opcija *auto_increment*. Ova kolona predstavlja primarni ključ za tabelu *Networks*.
- *NetworkName* – Naziv mreže.

Tabela *Nodes* ima sledeće kolone:

- *NodeID* – Jedinstveni ID čvora u vidu devetocifarnog broja. Ova kolona je primarni ključ tabele *Nodes.*
- *NetworkID* – ID mreže kojoj čvor pripada. Ova vrednost odgovara *NetworkID*-u iz tabele *Networks*
- *Xcoordinate* – X koordinata čvora
- *Ycoordinate* – Y koordinata čvora
- *NodeDelay* – Kašnjenje čvora

Tabela *Links* ima sledeće kolone:

- *LinkID* – Jedinstveni ID linka u vidu devetocifarnog broja. Ova kolona je primarni ključ tabele *Links*
- *NetworkID* – ID mreže kojoj link pripada. Odgovara *NetworkID*-u iz tabele *Networks*
- *LinkStartNode* – ID početnog čvora. Ova vrednost je jednaka odgovarajućem *NodeID*-u iz tabele *Nodes*
- *LinkEndNode* – ID krajnjeg čvora. Jednaka je odgovarajućem *NodeID*-u iz tabele *Nodes*
- *Price* – Cena linka
- *LinkCapacity* – Kapacitet linka
- *LinkDelay* – Kašnjenje linka

Kolone tabele *Flows* su:

- *FlowID* – Jedinstveni ID linka u vidu devetocifarnog broja. Ova kolona je primarni ključ tabele *Flows*
- *NetworkID* – ID mreže kojoj link pripada. Odgovara *NetworkID*-u iz tabele *Networks*
- *FlowStartNode* – ID početnog čvora. Ova vrednost odgovara *NodeID*-u iz tabele *Nodes*
- *FlowEndNode* – ID krajnjeg čvora. Odgovara *NodeID*-u iz tabele *Nodes*
- *FlowCapacity* – Kapacitet toka

Za kreiranje baze podataka sa prethodno navedenim tabelama, potrebno je pokrenuti sledeći .php file u brauzeru:

```
<?php
//povezivanje sa serverom
$connect = mysql_connect("localhost", "korisnik", "lozinka") or
     die ("Connection failed");
echo '<h3>'.'CREATE DATABASE IF NOT EXISTS topology DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci'.'</h3>';
//kreiraj novu bazu podataka - topology
$create = mysql_query("CREATE DATABASE IF NOT EXISTS topology DEFAULT CHARACTER
SET utf8 COLLATE utf8_general_ci")
  or die(mysql_error());

//selektujmo bazu topology za rad sa njom
mysql_select_db("topology");

//kreiranje tabela

//kreiranje tabele Networks
$Networks_table = "CREATE TABLE Networks (
  NetworkID int(9) unsigned NOT NULL auto_increment,
  NetworkName varchar(255) NOT NULL,
  PRIMARY KEY  (NetworkID)
)";
echo '<h3>'.$Networks_table.'</h3>';
//aktiviranje upita
$rezultat = mysql_query($Networks_table)
  or die (mysql_error());

//kreiranje tabele Nodes
  $NetworkNodes_table = "CREATE TABLE Nodes (
  NodeID int(9) unsigned NOT NULL auto_increment,
  NetworkID int(9) unsigned NOT NULL,
  Xcoordinate float(9,2) NOT NULL,
  Ycoordinate float(9,2) NOT NULL,
  NodeDelay int(9) unsigned NOT NULL,
  PRIMARY KEY  (NodeID)
)";
echo '<h3>'.$NetworkNodes_table.'</h3>';
//aktiviranje upita
$rezultat = mysql_query($NetworkNodes_table)
  or die (mysql_error());

//kreiranje tabele Links
$NetworkLinks_table = "CREATE TABLE Links (
  LinkID int(9) unsigned NOT NULL auto_increment,
  NetworkID int(9) unsigned NOT NULL,
  LinkStartNode int(9) unsigned NOT NULL,
  LinkEndNode int(9) unsigned NOT NULL,
  Price int(9) unsigned NOT NULL,
  LinkCapacity int(9) unsigned NOT NULL,
  LinkDelay int(9) unsigned NOT NULL,
  PRIMARY KEY  (LinkID)
)";
echo '<h3>'.$NetworkLinks_table.'</h3>';
//aktiviranje upita
$rezultat = mysql_query($NetworkLinks_table)
  or die (mysql_error());
```

```
//kreiranje tabele Flows
$NetworkFlows_table = "CREATE TABLE Flows (
  FlowID int(9) unsigned NOT NULL auto_increment,
  NetworkID int(9) unsigned NOT NULL,
  FlowStartNode int(9) unsigned NOT NULL,
  FlowEndNode int(9) unsigned NOT NULL,
  FlowCapacity int(9) unsigned NOT NULL,
  PRIMARY KEY  (FlowID)
)";
echo '<h3>'.$NetworkFlows_table.'</h3>';
//aktiviranje upita
$rezultat = mysql_query($NetworkFlows_table)
  or die (mysql_error());
//ispis poruke
echo "Baza uspešno kreirana";
?>
```

Da bi korisnik sačuvao u bazi podatke o mrežnoj topologiji koje je uneo preko korisničkog interfejsa, potrebno je da unese željeni naziv topologije i klikne na dugme “Upiši u bazu“. HTML kod kojim je određeno pomenuto dugme je sledeći:

```
<form>
Sacuvaj topologiju:</br>
Naziv topologije:
<input id="name"><button type="button" id="WriteDB">Upiši u bazu</button>
</form>
```

Ukoliko dati naziv topologije već postoji u bazi, korisnik će biti obavešten i moći će da izabere da li želi da unese topologiju sa drugim nazivom ili želi da ažurira topologiju koja već postoji u bazi. Ovo je realizovano *JavaScript* funkcijom koja se poziva klikom na dugme “Upiši u bazu“:

```
$("#WriteDB").click(function() {
     var name=document.getElementById("name").value;
     if (name==null || name=="") {
             alert('Unesite naziv topologije!')
             }
     else {
             jQuery.ajax({            //naziv mreze se salje u
"isTopologyInDB.php" gde se proverava da li vec postoji u bazi, ako ne postoji
vraca se "1"
             type: "POST",
             dataType: "text",
             url: "isTopologyInDB.php",
             data: name,
             success : function(data){
                if (data=="1")  {   //ako ime mreze ne postoji vec u bazi, uneti
podaci mogu da se upisu u bazu
                  var niz=new Array();
                  niz[0]=name;
                  niz[1]=Cvorovi;
                  niz[2]=Linkovi;
                  niz[3]=Tokovi;
                  jQuery.ajax({
                    type: "POST",
                    contentType: "application/json; charset=utf-8",
```

```
                    url: "sendToDB.php",
                    data: JSON.stringify(niz),
                    success : function(data){
                       alert('Podaci su uspesno ubaceni');
                       location.reload();
                    }
                   });
                }
                else {
                  r = confirm("Topologija sa datim nazivom već postoji u bazi!
Da li želite da je ažurirate?");
                  if (r==true) {
                    var niz=new Array();
                    niz[0]=name;
                    niz[1]=Cvorovi;
                    niz[2]=Linkovi;
                    niz[3]=Tokovi;
                    jQuery.ajax({
                      type: "POST",
                      contentType: "application/json; charset=utf-8",
                      url: "updateDB.php",
                      data: JSON.stringify(niz),
                      success : function(data){
                                  alert('Topologija je ažurirana!');
                                  location.reload();
                                }
                      });
                    }
                  }
              }
          });
    }
})
```

Na početku funkcije deklariše se promenljiva *name*, u koju se smešta sadržaj tekstualnog polja sa identifikatorom "*name*", odnosno naziv topologije. Ukoliko promenljiva *name* nije prazna, proverava se da li topologija sa tim nazivom već postoji u bazi. To se obavlja pomoću ***jQuery.ajax()*** metode. *AJAX* (*Asynchronous JavaScript and XML*) je tehnika za kreiranje brzih i dinamičkih veb strana, koja omogućava ažuriranje veb strana asinhrono, menjanjem male količine podataka dok se komunicira sa serverom, pri čemu se stranica ne mora ponovo učitati. Više detalja o tome kako *Ajax* funkcioniše se može pronaći na [11]. Biblioteka *jQuery* u potpunosti podržava *Ajax* i ima odličnu dokumentaciju koja se može pronaći na veb sajtu [12]. Metoda ***jQuery.ajax()*** izvršava asinhroni HTTP (*Ajax*) zahtev. Tip zahteva je definisan parametrom *type* i može biti POST, GET, PUT ili DELETE. U ovoj funkciji se koristi POST. Parametar *data* označava podatke koji se šalju ka serveru. Parametar URL je URL stranice ka kojoj se zahtev šalje, a *dataType* je tip podatka koji se očekuje kao odgovor od servera. Parametar *success* označava funkciju koja će se pozvati ako je zahtev uspešan. Ulazni parametar ove funkcije i parametar koji ona vraća je odgovor od servera, koji je formatiran u skladu sa definisanom vrednošću parametra *dataType*. Ideja je da se metoda ***jQuery.ajax()*** iskoristi za proveravanje postojanja topologije sa datim nazivom u bazi i upisivanje sadržaja nizova *Cvorovi*, *Linkovi* i *Tokovi* u bazu. Da li naziv topologije koji je korisnik uneo već postoji u bazi proverava se kodom na stranici isTopologyInDB.php. Ako vrednost definisana promenljivom *name*, koja se prosleđuje ovoj stranici, ne postoji u bazi, vratiće se vrednost "1", koja je definisana proizvoljno. Sadržaj stranice isTopologyInDB.php je sledeći:

```
<?php
```

```
  //čitanje podataka prosleđenih fajlu
  $name = file_get_contents("php://input");
  $output=true;

  //povezivanje sa serverom
  $connect = mysql_connect("localhost", "korisnik", "lozinka")
  or die ("Connection failed");

  $rezultat = mysql_query("SET NAMES 'utf8'")
    or die(mysql_error());

  //bira se baza topology za rad
  mysql_select_db("topology");
  $upit = "SET NAMES utf8";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //selektuju se svi zapisi iz kolone NetworkName iz tabele Networks
  $upit = "SELECT NetworkName from Networks";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //prolazi se kroz svaki zapis naziva topologije i proverava se da li je jednak
prosleđenoj vrednosti
  while ($zapis = mysql_fetch_assoc($rezultat)) {
    if ($zapis['NetworkName']==$name) {
      $output=false;
    }
  }

  //prosleđena vrednost ne postoji u bazi i stranica vraća vrednost "1"
  if ($output) {
    echo "1";
  }
  //prosleđena vrednost postoji u bazi i stranica vraća vrednost "0"
  else  {
    echo "0";
  }
?>
```

Podaci iz *Ajax* POST zahteva koji se prosleđuju stranici se čitaju pomoću *read-only stream*-a `php://input`, a sadržaj ovog fajla se pomoću funkcije ***file_get_contents()*** prosleđuje promenljivoj *$name*. Kao što se može videti iz navedenog koda, iz tabele *Networks* se selektuju svi zapisi iz kolone *NetworkName*, a zatim se za svaki zapis proverava da li je jednak nazivu topologije, sadržanom u promenljivoj $*name*. Ukoliko se pronađe takav zapis, promenljiva *$output* se postavlja na vrednost "*false*" i vraća se vrednost "0", a ako se ne pronađe, stranica vraća vrednost "1".

Za slučaj da je POST zahtev u metodi ***jQuery.ajax()*** bio uspešan i stranica isTopologyInDB.php vratila vrednost "1", znači da naziv topologije koji je korisnik uneo ne postoji već u bazi i da se sadržaj nizova *Cvorovi*, *Linkovi* i *Tokovi* može proslediti serveru i upisati u bazu. Ako je stranica isTopologyInDB.php vratila vrednost "0", znači da naziv topologije koji je korisnik uneo već postoji u bazi i korisniku se pruža mogućnost da izabere da li će da ažurira postojeću topologiju ili će da otkaže upisivanje topologije u bazu. Da bi se pojavio prozor za odabir jedne od ove dve mogućnosti, koristi se metoda ***confirm()***. Ulazni parametar ove metode je poruka koja će se prikazati korisniku, a izlazni parametar je vrednost "*true*", ako je korisnik kliknuo na "*OK*", odnosno "*false*", ako je korisnik kliknuo na "*Cancel*". U oba slučaja, kada se radi slanje podataka

ka serveru, definiše se promenljiva *niz*, sa 4 elementa. Ti elementi su promenljive *name*, *Cvorovi*, *Linkovi* i *Tokovi*. Promenljiva *niz* se zatim šalje ka serveru pomoću *Ajax* POST zahteva, pri čemu je format podatka koji se koristi *JSON* (JavaScript Object Notation). *JSON* format je sintaksa za čuvanje i razmenjivanje podataka, jednostavna za čitanje i pisanje ljudima, a takođe i mašinima za parsiranje i generisanje. Koristi se za serijalizaciju objekata, nizova, brojeva, *string*-ova, logičkih vrednosti i vrednosti *null*. Više o *JSON* formatu se može naći na veb sajtu [13]. Da bi se promenljiva *niz* prosledila php stranici, potrebno je serijalizovati niz pomoću metode ***JSON.stringify***, a zatim specificirati *contentType* Ajax zahteva, kako bi server razumeo da se radi o *JSON* formatu.

Za prosleđivanje nove topologije u bazu, koristi se deo koda sa str. 50, koji je naveden ispod i koji pripada php stranici sendToDB.php:

```
if (data=="1")  {
   var niz=new Array();
   niz[0]=name;
   niz[1]=Cvorovi;
   niz[2]=Linkovi;
   niz[3]=Tokovi;
         jQuery.ajax({
         type: "POST",
         contentType: "application/json; charset=utf-8",
         url: "sendToDB.php",
         data: JSON.stringify(niz),
         success : function(data){
         alert('Podaci su uspešno ubačeni');
         location.reload();
         }
       });
}
```

Kod stranice sendToDB.php je sledeći:

```
<?php
  //čitanje podataka prosleđenih fajlu
  $x=file_get_contents("php://input");
  $y=json_decode($x, true);
  echo $y[1][0]['KoordinataX'];
  echo count($y[1]);

  //povezivanje sa serverom
  $connect = mysql_connect("localhost", "korisnik", "lozinka")
    or die ("Connection failed");

  $rezultat = mysql_query("SET NAMES 'utf8'")
    or die(mysql_error());

  //bira se baza topology za rad
  mysql_select_db("topology");
  $upit = "SET NAMES utf8";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //upisivanje u tabelu Networks
  $ubaci = "INSERT INTO Networks (NetworkName) VALUES ('".$y[0]."')";
  $rezultat = mysql_query($ubaci)
```

```
    or die(mysql_error());

  //čita se NetworkID topologije koja je upravo upisana
  $upit = "SELECT NetworkID from Networks WHERE NetworkName='".$y[0]."'";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  $row=mysql_fetch_array($rezultat);
  $id=$row['NetworkID'];

  //određuje se broj čvorova
  $br=count($y[1]);
  //upisivanje svakog čvora u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Nodes (NetworkID, Xcoordinate, Ycoordinate, NodeDelay)
    VALUES ('".$id."',
'".$y[1][$m]['KoordinataX']."','".$y[1][$m]['KoordinataY']."','".$y[1][$m]['Kasn
jenjeCvora']."')";
    $rezultat = mysql_query($ubaci)
     or die(mysql_error());
  }

  //određuje se broj linkova
  $br=count($y[2]);
  //upisivanje svakog linka u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Links (NetworkID, LinkStartNode, LinkEndNode, Price,
LinkCapacity, LinkDelay)
    VALUES
('".$id."','".$y[2][$m]['PocetniCvor']."','".$y[2][$m]['KrajnjiCvor']."','".$y[2
][$m]['Cena']."','".$y[2][$m]['Kapacitet']."','".$y[2][$m]['KasnjenjeLinka']."')
";
    $rezultat = mysql_query($ubaci)
        or die(mysql_error());
  }

  //određuje se broj tokova
  $br=count($y[3]);
  //upisivanje svakog toka u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Flows (NetworkID,FlowStartNode, FlowEndNode,
FlowCapacity)
    VALUES
('".$id."','".$y[3][$m]['PocetniCvorToka']."','".$y[3][$m]['ZavrsniCvorToka']."'
,'".$y[3][$m]['KapacitetToka']."')";
    $rezultat = mysql_query($ubaci)
        or die(mysql_error());
  }
?>
```

U fajlu sendToDB.php se prvo definiše promenljiva *$x*, u koju se smešta sadržaj tela *Ajax* POST zahteva, odnosno promenljiva niz u *JSON* formatu, koja se zatim konvertuje u PHP promenljivu *$y*, pomoću metode ***json_decode()***. Dobijena promenljiva *$y* je niz čiji je prvi element naziv topologije koja je će se upisati u bazu kao *NetworkName*, a drugi, treći i četvrti elementi su nizovi objekata *Cvorovi*, *Linkovi* i *Tokovi*, respektivno. Nakon ovoga sledi upisivanje naziva nove topologije u bazu, i čitanje ID-a upravo upisanog naziva topologije u tabelu *Networks*, jer se prilikom upisivanja čvorova, linkova i tokova zahteva i ID topologije kojoj čvor/link/tok pripada. ID topologije,

odnosno zapis iz kolone *NetworkID,* se smešta u promenljivu *$id*. Zatim se za svaki od preostala tri elementa niza *$y* određuje dužina, odnosno određuju se dužine nizova *Cvorovi*, *Linkovi* i *Tokovi*, koji predstavljaju ova tri elementa, da bi se zatim pomoću WHILE petlje parametri svakog čvora, linka i toka upisali u tabele *Nodes*, *Links* i *Flows*, respektivno. Na taj način je završen upis u bazu topologije sa novim nazivom. Izvršava se *success* funkcija *Ajax* POST zahteva, kada korisnik dobija poruku "Podaci su uspešno bačeni!".

Za upisivanje topologije koja sa navedenim nazivom već postoji u bazi, odnosno za ažuriranje topologije, izvršava se deo koda funkcije sa str. 70, koja se poziva klikom na dugme "Upiši u bazu":

```
else {
                    r = confirm("Topologija sa datim nazivom već postoji u bazi!
Da li želite da je ažurirate?");
                    if (r==true) {
                      var niz=new Array();
                      niz[0]=name;
                      niz[1]=Cvorovi;
                      niz[2]=Linkovi;
                      niz[3]=Tokovi;
                      jQuery.ajax({
                        type: "POST",
                        contentType: "application/json; charset=utf-8",
                        url: "updateDB.php",
                        data: JSON.stringify(niz),
                        success : function(data){
                                    alert('Topologija je ažurirana!');
                                    location.reload();
                                  }
                        });
                      }
                    }
                }
        });
    }
})
```

Kao što se može primetiti, promenljiva niz se pomoću metode ***JSON.stringify()*** konvertuje u *JSON* string i prosleđuje fajlu updateDB.php. Stranica updateDB.php, na isti način kao i stranica sendToDB.php, prihvata sadržaj promenljive niz pomoću funkcije ***file_get_contents()*** i pridružuje ga PHP promenljivoj *$y*. Naime, stranica updateDB.php sadrži sledeći kod:

```
<?php
  //čitanje podataka prosleđenih fajlu
  $x=file_get_contents("php://input");
  $y=json_decode($x, true);

  //povezivanje sa serverom
  $connect = mysql_connect("localhost", "korisnik", "lozinka")
  or die ("Connection failed");

  $rezultat = mysql_query("SET NAMES 'utf8'")
    or die(mysql_error());

  //biramo bazu topology
  mysql_select_db("topology");
```

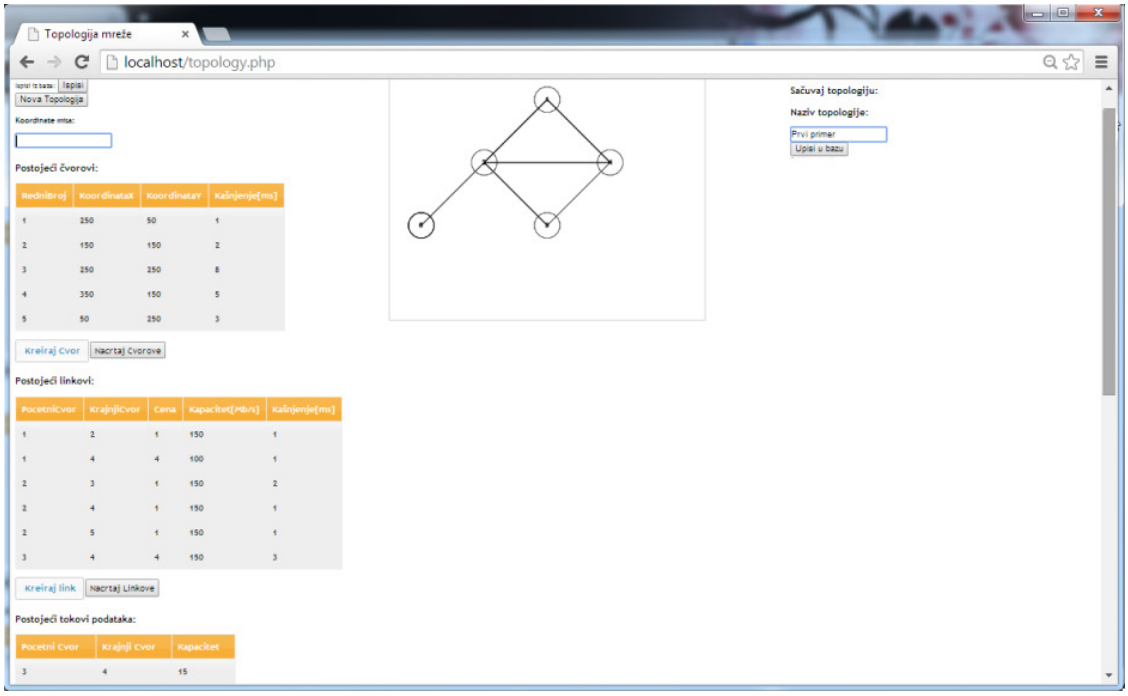
```
  //selektovanje NetworkID iz tabele Networks sa prosleđenim nazivom topologije
  $upit = "SELECT NetworkID from Networks WHERE NetworkName='".$y[0]."'";
  $rezultat = mysql_query($upit)
    or die(mysql_error());
  $row=mysql_fetch_array($rezultat);
  $id=$row['NetworkID'];

  //brisanje iz tabele Nodes svih zapisa vezanih za prosleđeni naziv topologije
  $upit = "DELETE FROM Nodes WHERE NetworkID=$id";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //brisanje iz tabele Links svih zapisa vezanih za prosleđeni naziv topologije
  $upit = "DELETE FROM Links WHERE NetworkID=$id";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //brisanje iz tabele Flows svih zapisa vezanih za prosleđeni naziv topologije
  $upit = "DELETE FROM Flows WHERE NetworkID=$id";
  $rezultat = mysql_query($upit)
    or die(mysql_error());

  //određuje se broj čvorova
  $br=count($y[1]);
  //upisivanje svakog čvora u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Nodes (NetworkID, Xcoordinate,Ycoordinate,NodeDelay)
    VALUES ('".$id."',
'".$y[1][$m]['KoordinataX']."','".$y[1][$m]['KoordinataY']."','".$y[1][$m]['Kasn
jenjeCvora']."')";
    $rezultat = mysql_query($ubaci)
     or die(mysql_error());
  }

  //određuje se broj linkova
  $br=count($y[2]);
  //upisivanje svakog linka u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Links (NetworkID, LinkStartNode, LinkEndNode, Price,
LinkCapacity, LinkDelay)
    VALUES
('".$id."','".$y[2][$m]['PocetniCvor']."','".$y[2][$m]['KrajnjiCvor']."','".$y[2
][$m]['Cena']."','".$y[2][$m]['Kapacitet']."','".$y[2][$m]['KasnjenjeLinka']."')
";
    $rezultat = mysql_query($ubaci)
        or die(mysql_error());
  }

  //određuje se broj tokova
  $br=count($y[3]);
  //upisivanje svakog toka u bazu
  for ($m=0; $m<$br; $m++)  {
    $ubaci = "INSERT INTO Flows (NetworkID,FlowStartNode, FlowEndNode,
FlowCapacity)
    VALUES
('".$id."','".$y[3][$m]['PocetniCvorToka']."','".$y[3][$m]['ZavrsniCvorToka']."'
,'".$y[3][$m]['KapacitetToka']."')";
    $rezultat = mysql_query($ubaci)
        or die(mysql_error());
```

```
    }
?>
```

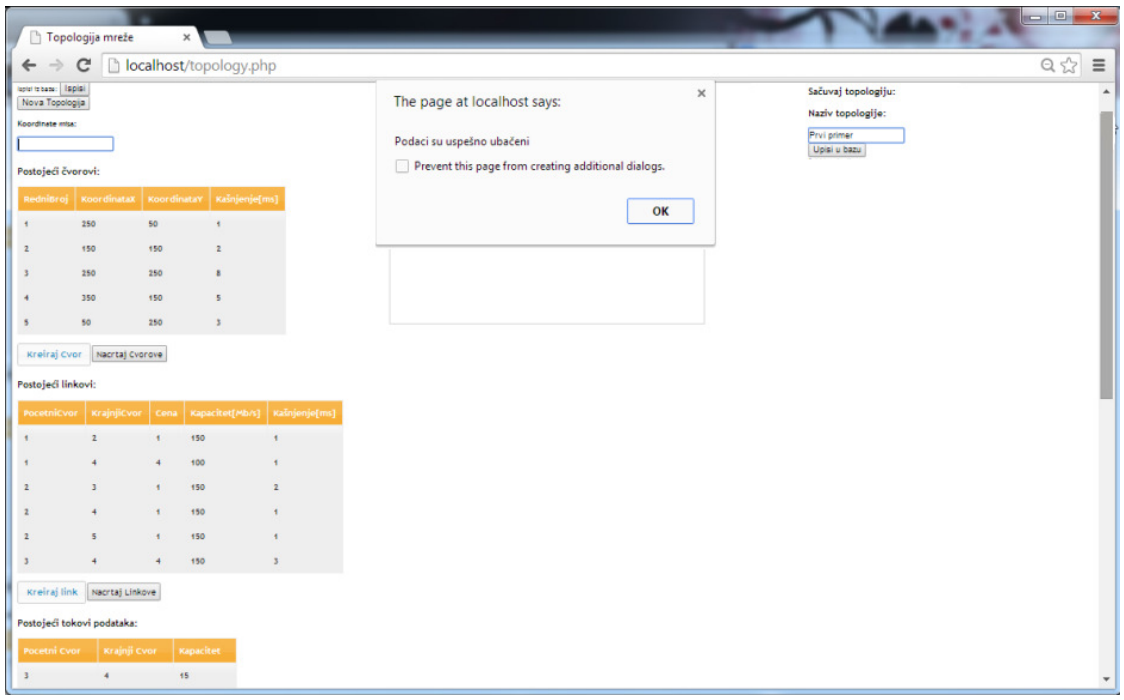
Nakon što se odabere baza *topology*, iz tabele *Networks* selektuje se *NetworkID* koji odgovara prosleđenom nazivu topologije, odnosno prvom elementu niza *$y*. To se radi da bi se iz tabela *Nodes*, *Links* i *Flows* obrisali svi redovi koji imaju taj NetworkID, posle čega se u te tabele vrši upis elemenata nizova *Cvorovi*, *Linkovi* i *Tokovi*, sa ID-em mreže koji je jednak vrednosti za *NetworkID*. Upis se vrši na isti način kao kod dodavanja topologije sa nazivom koji ne postoji u bazi – određuje se broj čvorova, linkova i tokova i za svaki element se vrši upis njegovih parametara u odgovarajuću tabelu. Kada se završi sa upisom u bazi, poziva se *success* funkcija *Ajax* POST requesta koja obaveštava korisnika da je topologija ažurirana.

Na slici 5.5.1 prikazan je primer upisivanja kreirane mrežne topologije u bazu. Pošto su uneti elementi mrežne topologije, korisnik treba da unese naziv topologije u tekstualno polje i klikne na dugme Upisi u bazu, a nakon upisa u bazu pojaviće se obaveštenje kao na slici 5.5.2, da su podaci uspešno ubačeni. Na kraju se stranica ponovo učitava i u listi sačuvanih topologija, se pojavljuje naziv upravo sačuvane topologije sa nazivom "Prvi primer", kao na slici 5.5.3. Više o ispisivanju naziva topologije može se pročitati u sledećem odeljku.

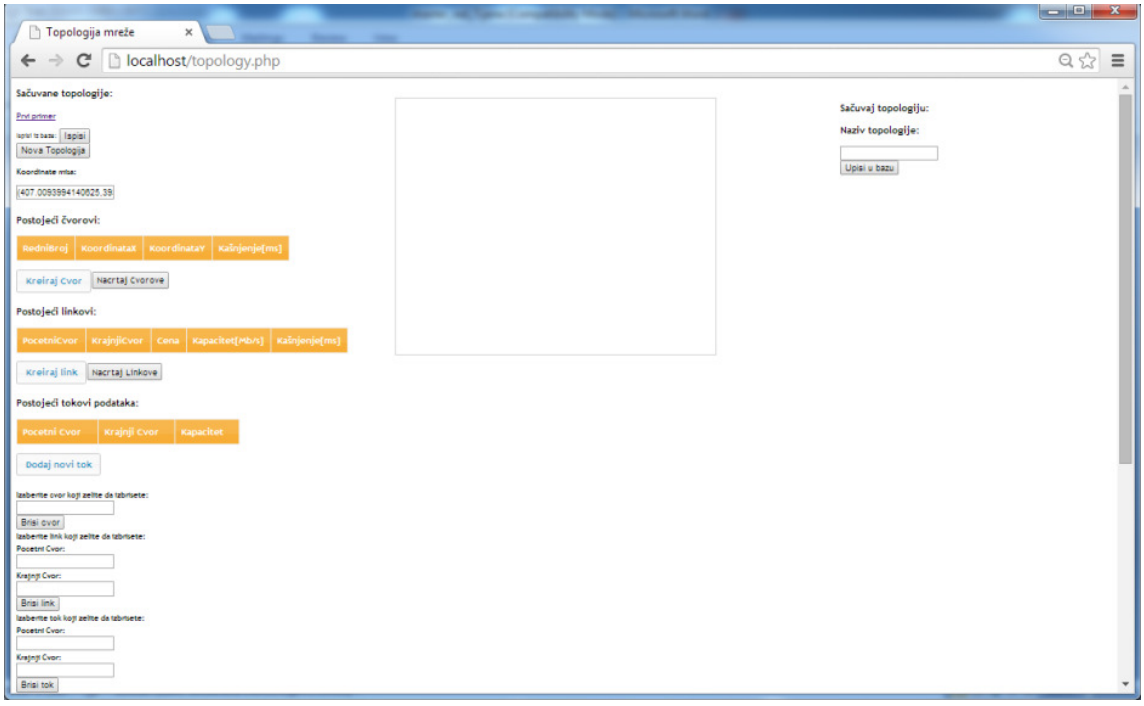
Ako korisnik unese novu topologiju i želi da je sačuva takođe pod nazivom "Prvi primer", pojaviće se dijalog kao na slici 5.5.4. Ako se klikne na OK, podaci o topologiji "Prvi primer", sa slike 5.5.1 će se izbrisati iz baze, a pod tim nazivom će biti upisani podaci o novokreiranoj topologiji. Ako se klikne na "*Cancel*", upisivanje u bazu se neće obaviti, a korisnik će moći da unese neki drugi naziv za naziv topologije.



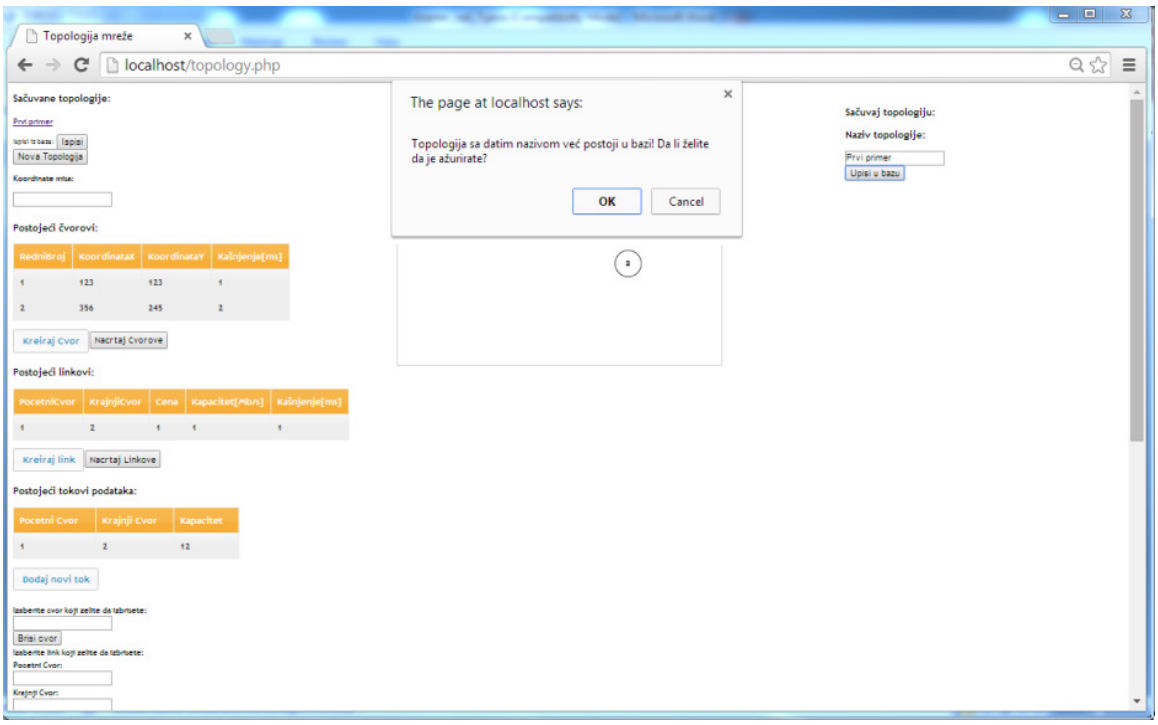
Slika 5.5.1 Upisivanje topologije u bazu



Slika 5.5.2. Podaci su uspešno upisani u bazu



Slika 5.5.3. Sačuvana topologija se pojavljuje u listi sačuvanih topologija na stranici



Slika 5.5.4. Upisivanje topologije u bazu čiji naziv već postoji u bazi

5.6. Ispisivanje topologije iz baze podataka

U ovom odeljku je opisano čitanje iz baze i ispisivanje postojećih topologija na veb stranici. Kao što je opisano u prethodnom odeljku i prikazano na slici 5.5.3, nazivi topologija su ispisani na stranici u vidu linkova. To je postignuto korišćenjem sledećeg koda:

```
<?php
//povezivanje sa serverom
$connect = mysql_connect("localhost", "korisnik", "lozinka")
   or die ("Connection failed");

mysql_query("SET NAMES 'utf8'")  or die(mysql_error());

//biramo bazu topology
mysql_select_db("topology");

//selektuju se svi nazivi topologija iz tabele Networks
$upit = "SELECT NetworkName from Networks";
$rezultat = mysql_query($upit)
   or die(mysql_error());

//ispisivanje linkova sa nazivima topologija iz baze
while ($zapis = mysql_fetch_assoc($rezultat)) {
   echo "<h3><a
href='topology.php?top=".$zapis['NetworkName']."'>".$zapis['NetworkName']."</a><
/h3>";
        }
```

Ako korisnik klikne na link željene topologije, naziv te topologije, koji se nalazi u koloni *NetworkName*, biće prosleđen topology.php stranici kroz parametar *top*, što je definisano URL-om

u poslednjem redu koda prikazanog iznad. Vrednosti promenljive *top* se pristupa pomoću superglobalne php promenljive *$_GET* i prosleđena vrednost promenljive *top* se smešta u pomoćnu promenljivu *$_TopologyName*, radi lakšeg razumevanja koda. Nakon toga se iz tabela *Nodes*, *Links* i *Flows* selektuju svi čvorovi, linkovi i tokovi koji pripadaju topologiji sa nazivom *$_TopologyName*. Svaki selektovani čvor se dodaje u multidimenzionalni niz *$Nodes* kao novi element, pri čemu je svaki element niza *$Nodes* niz, čiji indeksi nose nazive parametara čvora i shodno tim nazivima se popunjavaju elementi elemenata niza *$Nodes*. PHP promenljiva *$Nodes* je ekvivalent *JavaScript* promenljive *Cvorovi*. Analogno nizu *$Nodes* kreiraju se nizovi *$Links* i *$Flows*. Opisane akcije su realizovane kodom prikazanim ispod.

```
//u promenljivu $TopologyName se smešta naziv topologije na čiji link je
kliknuto
if (isset($_GET['top'])) {
            $TopologyName=$_GET['top'];

            //selektovanje svih podataka iz tabele Nodes koji pripadaju
topologiji $TopologyName
            $upit = "SELECT Nodes.Xcoordinate,
Nodes.Ycoordinate,Nodes.NodeDelay,Networks.NetworkName from Nodes LEFT JOIN
Networks ON Nodes.NetworkID=Networks.NetworkID WHERE
Networks.NetworkName='".$TopologyName."'";
            $rezultat = mysql_query($upit)
            or die(mysql_error());
            $k=0;
            while ($zapis = mysql_fetch_assoc($rezultat)) {
//php niz ekvivalentan javascript nizu Cvorovi
$Nodes[$k]=array("KoordinataX"=>$zapis['Xcoordinate'],"KoordinataY"=>$zapis['Yco
ordinate'],"KasnjenjeCvora"=>$zapis['NodeDelay']);
                $k++;
            }

            //selektovanje svih podataka iz tabele Links koji pripadaju
topologiji $TopologyName
            $upit = "SELECT Links.LinkStartNode, Links.LinkEndNode, Links.Price,
Links.LinkCapacity, Links.LinkDelay from Links LEFT JOIN Networks ON
Links.NetworkID=Networks.NetworkID WHERE
Networks.NetworkName='".$TopologyName."'";
            $rezultat = mysql_query($upit)
            or die(mysql_error());
            $k=0;
            while ($zapis = mysql_fetch_assoc($rezultat)) {
//php niz ekvivalentan javascript nizu Linkovi
$Links[$k]=array("PocetniCvor"=>$zapis['LinkStartNode'],"KrajnjiCvor"=>$zapis['L
inkEndNode'],"Cena"=>$zapis['Price'],"Kapacitet"=>$zapis['LinkCapacity'],"Kasnje
njeLinka"=>$zapis['LinkDelay']);
                $k++;
            }

            //selektovanje svih podataka iz tabele Flows koji pripadaju
topologiji $TopologyName
            $upit = "SELECT Flows.FlowStartNode, Flows.FlowEndNode,
Flows.FlowCapacity,Networks.NetworkName from Flows LEFT JOIN Networks ON
Flows.NetworkID=Networks.NetworkID WHERE
Networks.NetworkName='".$TopologyName."'";
            $rezultat = mysql_query($upit)
            or die(mysql_error());
```

```
            $k=0;
            while ($zapis = mysql_fetch_assoc($rezultat)) {
//php niz ekvivalentan javascript nizu Tokovi
$Flows[$k]=array("PocetniCvorToka"=>$zapis['FlowStartNode'],"ZavrsniCvorToka"=>$
zapis['FlowEndNode'],"KapacitetToka"=>$zapis['FlowCapacity']);
                $k++;
            }


        }
    ?>
```

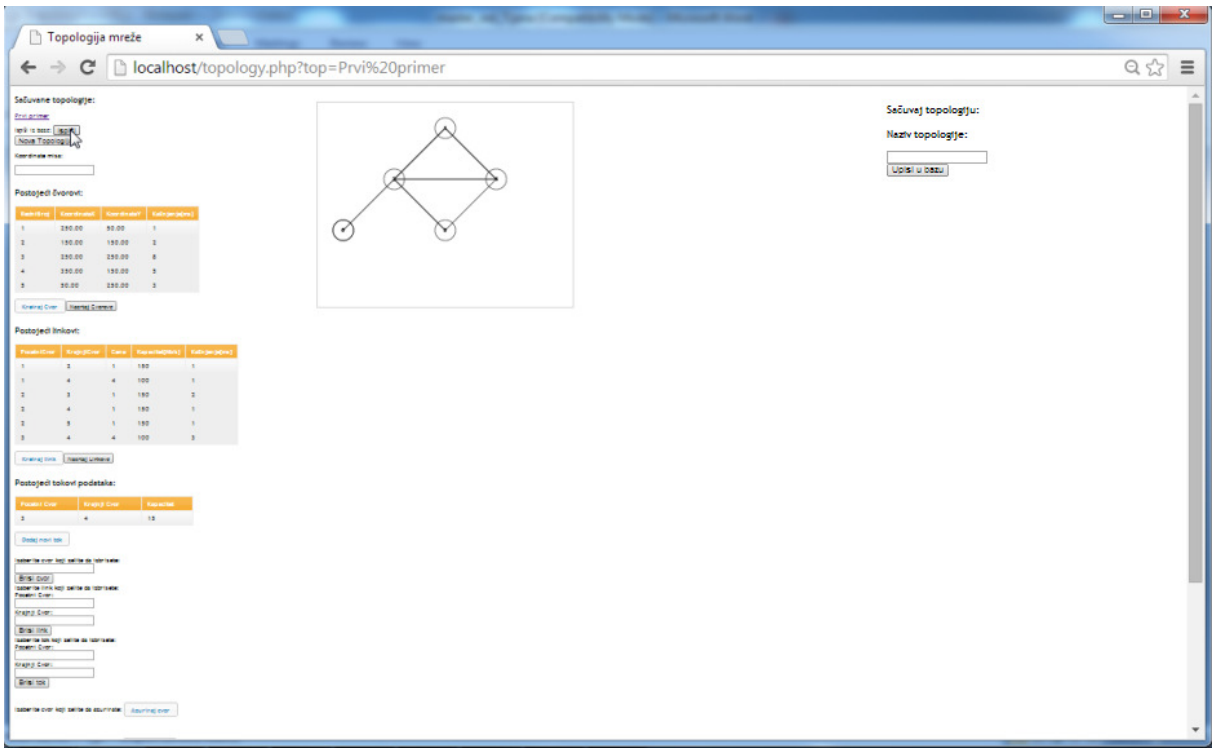
Da bi se topologija iscrtala na topology.php stranici i da bi se dodavali, ažurirali, brisali i čuvali novi parametri čvorova, linkova i tokova, pomoću opisanih *JavaScript* funkcija, vrednosti php promenljivih *$Nodes*, *$Links* i *$Flows* se moraju proslediti promenljivama *Cvorovi*, *Linkovi* i *Tokovi*, respektivno. To se jednostavno može uraditi pomoću *JSON* formata, na sledeći način:

```
<script>
//dodela sadržaja php promenljive javascript promenljivoj
obj = JSON.parse('<?php echo json_encode($Nodes) ?>');
Cvorovi=obj;
obj = JSON.parse('<?php echo json_encode($Links) ?>');
Linkovi=obj;
obj = JSON.parse('<?php echo json_encode($Flows) ?>');
Tokovi=obj;
CrtajSve(Cvorovi,Linkovi,Tokovi);
</script>
```

Metoda ***json_encode()***, suprotno od metode ***json_decode()***, vraća vrednost PHP promenljive u JSON format. Metoda ***JSON.parse***, suprotno od metode ***JSON.stringify***, parsira *JSON* string. Posle konverzije, preostaje samo prikazivanje parametara elemenata mreže i crtanje topologije u elementu *canvas*, za šta je potrebno da korisnik klikne na dugme “Ispiši“:

```
<form>
Ispiši iz baze:
<button type="button" id="ReadDB">Ispiši</button>
</form>
```

Klikom na ovo dugme se poziva funkcija ***CrtajSve()***. Na slici 5.6.1 prikazano je crtanje topologije i njenih parametara sačuvane u bazi pod nazivom “Prvi primer“.



5.6.1. Klikom na dugme Ispiši ispisuje se sadržaj iz baze o izabrane topologije

5.7. Brisanje sačuvane topologije iz baze

Brisanje topologije iz baze se obavlja upisivanjem naziva topologije, koja se želi izbrisati, u tekstualno polje i klikom na dugme "Briši", čiji je HTML kod prikazan ispod.

```
<form>
Izbrišite topologiju iz baze:</br>
Naziv topologije: <input type="text" id="delete_name"><button type="button"
id="DeleteDB">Briši</button>
</form>
```

Klikom na dato dugme se poziva *jQuery* funkcija u kojoj se vrednost tekstualnog polja *delete_name* dodeljuje promenljivoj *name*. Promenljiva *name* se zatim prosleđuje stranici delete.php pomoću ***jQuery.ajax()*** metode:

```
$("#DeleteDB").click(function(){
  var name=document.getElementById("delete_name").value;
  jQuery.ajax({
    type: "POST",
    dataType: "text",
    url: "delete.php",
    data: name,
    success : function(data){
    alert(data);
    location.reload();
   }
 })
```

```
})
```

Na stranici delete.php, prosleđena vrednost se smešta u promenljivu *$name* i iz tabele *Networks* selektuje se *NetworkID* koji odgovara nazivu topologije *$name*. Zapis *NetworkID* se dodeljuje promenljivoj *$id*:

```
<?php
//čitanje podataka prosleđenih fajlu
$name = file_get_contents("php://input");
$connect = mysql_connect("localhost", "korisnik", "lozinka")
  or die ("Connection failed");

//biramo bazu topology
mysql_select_db("topology");

$upit = "SELECT NetworkID from Networks WHERE NetworkName='".$name."'";
$rezultat = mysql_query($upit)
    or die(mysql_error());

$row=mysql_fetch_array($rezultat);
$id=$row['NetworkID'];
```

Nakon ovoga, pomoću SQL komande DELETE lako je obrisati sve zapise iz tabela *Nodes*, *Links* i *Flows*, čija je vrednost kolone *NetworkID* jednala vrednosti promenljive *$id*, identifikatoru topologije sa nazivom *$name*, što prikazuju sledeći redovi koda:

```
//brisanje zapisa iz tabele Nodes
$brisi = "DELETE FROM Nodes WHERE NetworkID='".$id."'";
$rezultat = mysql_query($brisi)
  or die(mysql_error());

//brisanje zapisa iz tabele Links
$brisi = "DELETE FROM Links WHERE NetworkID='".$id."'";
$rezultat = mysql_query($brisi)
    or die(mysql_error());

//brisanje zapisa iz tabele Flows
$brisi = "DELETE FROM Flows WHERE NetworkID='".$id."'";
$rezultat = mysql_query($brisi)
    or die(mysql_error());

//brisanje zapisa iz tabele Networks
$brisi = "DELETE FROM Networks WHERE NetworkID='".$id."'";
$rezultat = mysql_query($brisi)
    or die(mysql_error());

    echo "Topologija je izbrisana"
?>
```

Ukoliko se nije dogodila greška prilikom izvršavanja koda sa stranice delete.php, stranica vraća obaveštenje “Topologija je izbrisana“ i ponovo se učitava, pri čemu se u listi sačuvanih topologija može uočiti da topologije za brisanje više nema i da je zaista izbrisana.

6. ZAKLJUČAK

U ovom radu kreirana je i opisana biblioteka za dinamičko kreiranje i modifikovanje mrežne topologije primenom HTML5 elementa *canvas*.

JavaScript funkcije, koje su obuhvaćene bibliotekom i opisane u četvrtom poglavlju, omogućavaju dodavanje, brisanje i ažuriranje čvorova, linkova i tokova korisnika. Parametri čvora su koordinate u elementu *canvas* i kašnjenje čvora. Parametri linka su njegovi krajnji čvorovi, cena, kapacitet i kašnjenje, a parametri toka korisnika su krajnji čvorovi i kapacitet toka. Za svaki element mrežne topologije se vrlo jednostavno može proširiti lista parametara. U petom poglavlju je pokazano da se data biblioteka može koristiti za grafički prikaz i modifikovanje mrežne topologije u elementu *canvas*, za određivanje optimalne putanje od odabranog čvora do svakog od čvorova u mreži primenom Dijkstra algoritma i da se mrežna topologija, primenom *Ajax* tehnike i *JSON* notacije, može sačuvati u bazi i kasnije čitati iz nje.

Integracijom *JavaScript* fajla sa funkcijama opisanim u četvrtom poglavlju, korisnik lako može uključiti kreiranu biblioteku u svoj veb sajt. Biblioteka se može koristi u različite svrhe, kao što su edukativne, istraživačke, kao i za onlajn demonstraciju rezultata istraživanja, odnosno rada razvijenih algoritama. Korisniku se pruža mogućnost da ima kompletan uvid u distribuciju uređaja na OSI sloju 2 ili 3, njihovu mrežnu povezanost i kapacitetu linkova između njih. Planiranje mrežne topologije i održavanje dokumentacije je olakšano grafičkim prikazom lokacija mrežnih elemenata i njihovih parametara.

Budući razvoj biblioteke išao bi u pravcu implementacije automatskog generisanja topologije, kao i dodavanja novih mrežnih elemenata na sloju 2 ili 3. Takođe, uvođenjem *drag and drop* funkcionalnosti u biblioteku, olakšala bi se modifikacija grafičkog prikaza mrežne topologije. Dodatno poboljšanje u vidu sigurnosti i lakšeg pristupa donelo bi korišćenje multi-korisničkog veb interfejsa, što bi značilo da bi se kontrolisao pristup nadgledanju i/ili modifikaciji različitih mrežnih topologija.

LITERATURA

[1] Aleksandra Smiljanić, predavanja iz predmeta Internet programiranje, Elektrotehnički fakultet, Beograd, 2012.

[2] Jeremy Keith, *HTML5 For Web Designers*, A Book Apart Jeffrey Zeldmann, 2010.

[3] Adam MCDaniel, *HTML5: Your visual blueprint for designing rich web pages and applications*, Visual, 2012.

[4] Anselm Bradford, Paul Haine, *HTML5 Mastery, Semantics, Standards and Styling*, Apress, 2011.

[5] Eric Rowell, *HTML5 Canvas Cookbook*, Packt Publishing, 2011.

[6] Rob Hawkes, *Foundation HTML5 Canvas For Games and Entertainment*, Apress, 2011.

[7] JavaScript Array splice() Method, http://www.w3schools.com/jsref/jsref_splice.asp

[8] Downloading jQuery, http://jquery.com/download/

[9] JavaScript HTML DOM Event Listener, http://www.w3schools.com/js/js_htmldom_eventlistener.asp

[10] Dialog, http://jqueryui.com/dialog/#modal-form

[11] Ajax Tutorial, http://www.w3schools.com/ajax/default.ASP

[12] jQuery API Documentation: Ajax, http://api.jquery.com/category/ajax/

[13] JSON, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

A. PRIMER *DRAG AND DROP* FUNKCIONALNOSTI

U ovom prilogu objašnjen je kod kojim su generisane slike koje su opisuju *drag and drop* funkcionalnost, u odeljku 2.2.7. Da bi se kreirale tri mete u koje će elementi biti ispušteni upotrebiće se <*div*> elementi i sledeći kod:

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<div id="target1"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
<div id="target2"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
<div id="target3"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
</body>
</html>
```

Kod u nastavku kreira tri <*div*> elementa sa atributom *draggable*, koja su podelementi prve mete.

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<h1> "Drag and Drop" Primer</h1>
<div id="target1"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
<div id="draggable1" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">1
</div>
<div id="draggable2" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">2
</div>
<div id="draggable3" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">3
</div>
```

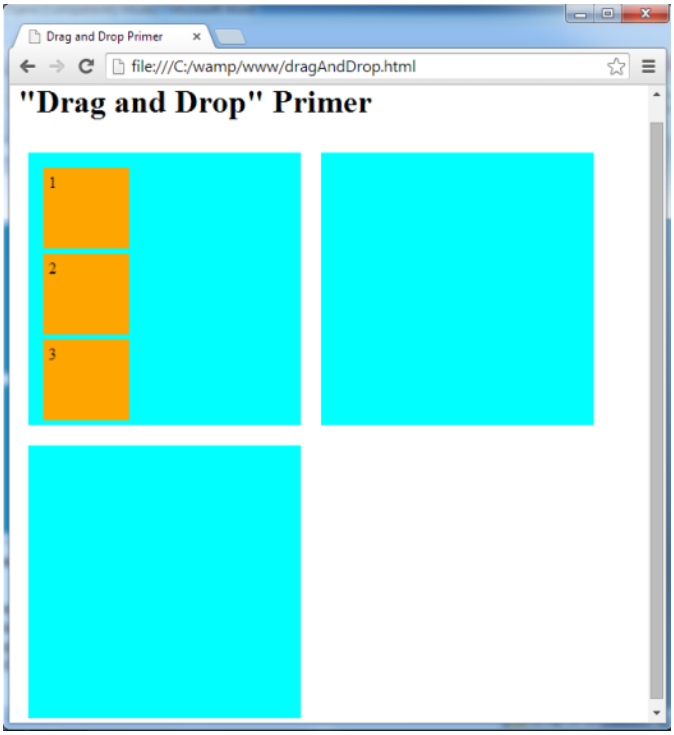
```
</div>

<div id="target2"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>

<div id="target3"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
</body>
</html>
```

U ovom koraku će * * elementi biti stilizovani. Mete će biti označene plavom, a pomerajući elementi narandžastom bojom.

```
<!DOCTYPE HTML>
<html>
<head>
<title>
Drag and Drop Primer
</title>
<style type="text/css">
#target1, #target2, #target3
{
float:left; width:250px; height:250px;
padding:10px; margin:10px;
}
#draggable1, #draggable2, #draggable3
{
width:75px; height:70px; padding:5px;
margin:5px;
}
#target1 {background-color: cyan;}
#target2 {background-color: cyan;}
#target3 {background-color: cyan;}
#draggable1 {background-color: orange;}
#draggable2 {background-color: orange;}
#draggable3 {background-color: orange;}
</style>
</head>
<body>
<h1> "Drag and Drop" Primer</h1>
<div id="target1"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
<div id="draggable1" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">1
</div>
<div id="draggable2" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">2
</div>
```

```
<div id="draggable3" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">3
</div>
</div>
<div id="target2"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
<div id="target3"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
</body>
</html>
```

Kada je pokrene fajl sa navedenim HTML kodom, stranica u brauzeru *Google Chrome* izgleda kao na slici B.1.



Slika B.1. Prikaz koda u brauzeru za primer *drag and drop* operacije

Kada korisnik počne da pomera pomerajući <*div*> element, inicira se *ondragstart* događaj, kojim se poziva funkcija koja će se u primeru nazvati ***start()***. U funkciji je potrebno postaviti da je pomerajući <*div*> element koji korisnik pokušava da premesti zaista pomerajući. To se postiže podešavanjem *dataTransfer.effectAllowed,* objekta koji je prosleđen ***start()*** funkciji, na događaj *move*:

```
<script type="text/javascript">
function start(e)
```

```
{
e.dataTransfer.effectAllowed='move';
.
.
.
}
```

Da bi se sačuvao *ID* <*div*> elementa koji se pomera i da bi se mogao pomerati i nakon što je prevučen, potrebno je dodati sledeći kod:

```
<script type="text/javascript">
function start(e)
{
e.dataTransfer.effectAllowed='move';
e.dataTransfer.setData("Data",
e.target.getAttribute('id'));
.
.
.
}
```

Kada korisnik prevuče pomerajući <*div*> element u ciljani <*div*> element, dešava se *ondragevent* događaj. Taj događaj inicira funkciju koja je u primeru nazvana ***enter()***. Ova funkcija vraća vrednost "*true*" kao indikaciju da je elementu koji se pomera dozvoljeno da se stupi u prostor mete.

```
function enter(e)
{
return true;
}
```

Kada korisnik prevlači element preko mete, dešava se *ondragover* događaj, koji inicira funkciju koja se u primeru zove ***over()***. Ova funkcija vraća vrednost "*true*" ako je objektu koji se pomera dozvoljeno da bude ispušten na datu metu, odnosno vraća vrednost "*false*" ako se objekat ne može prevući na datu metu.

```
function over(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
var id = e.target.getAttribute('id');
.
.
.
}
```

Dodati sledeći kod funkciji ***over()*** da bi se označilo da bilo koji element koji može da se pomera može biti prevučen na metu 1, da se <*div*> 1 može prevući samo na metu 2, a da se pomerajući <*div*> 1 i <*div*> 2 mogu pomeriti samo na metu 3.

```
function over(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
var id = e.target.getAttribute('id');
```

```
if(id ='target1')
return false;
if((id ='target2')
&& iddraggable == `draggable3')
return false;
else if(id ='target3'
&& (iddraggable == `draggable1' ||
iddraggable ='draggable2'))
return false;
else
return true;
}
```

Kada korisnik spusti pomerajući <*div*> element u prostor u koji mu je to dozvoljeno, poziva se *JavaScript* funkcija ***appendChild*** koja će pripojiti pomereni <*div*> element meti u koju je pomeren. U isto vreme dešava se *ondrop* događaj i poziva se ***drop()*** funkcija i dobija se *ID* pomerenog elementa.

```
function drop(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
.
.
.
}

<script/>
```

Da bi se element koji se pomerio bio pripojen meti i da bi se prekinula dalja propagacija događaja u brauzeru funkcijom ***stopPropagation***, dodaje se sledeći kod:

```
function drop(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
e.target.appendChild
(document.getElementById(iddraggable));
e.stopPropagation();
return false;
}
```

Na kraju ostaje *ondragend* događaj, kojim se poziva funkcija ***end()*** i kojom će se osloboditi podaci sačuvani u *dataTransfer* objektu (*ID* elementa koji se pomerao) i time će *drop* operacija biti završena.

```
function end(e)
{
e.dataTransfer.clearData("Data");
return true
}
</script>
```

Kompletan kod primera:

```
<!DOCTYPE HTML>
<html>
<head>
<title>
Drag and Drop Primer
</title>
<style type="text/css">
#target1, #target2, #target3
{
float:left; width:250px; height:250px;
padding:10px; margin:10px;
}
#draggable1, #draggable2, #draggable3
{
width:75px; height:70px; padding:5px;
margin:5px;
}
#target1 {background-color: cyan;}
#target2 {background-color: cyan;}
#target3 {background-color: cyan;}
#draggable1 {background-color: orange;}
#draggable2 {background-color: orange;}
#draggable3 {background-color: orange;}
</style>
<script type="text/javascript">
function start(e)
{
e.dataTransfer.effectAllowed='move';
e.dataTransfer.setData("Data",
e.target.getAttribute('id'));
//e.dataTransfer.setDragImage(e.target, 0, 0);
return true;
}
function enter(e)
{
return true;
}
function over(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
var id = e.target.getAttribute('id');
if(id =='target1')
return false;
if((id =='target2')
&& iddraggable == 'draggable3')
return false;
else if(id =='target3'
&& (iddraggable == 'draggable1' ||
iddraggable =='draggable2'))
return false;
else
return true;
}
function drop(e)
{
var iddraggable =
e.dataTransfer.getData("Data");
e.target.appendChild
```

```
(document.getElementById(iddraggable));
e.stopPropagation();
return false;
}
function end(e)
{
e.dataTransfer.clearData("Data");
return true
}
</script>
</head>
<body>
<h1> "Drag and Drop" Primer</h1>
<div id="target1"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
<div id="draggable1" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">1
</div>
<div id="draggable2" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">2
</div>
<div id="draggable3" draggable="true"
ondragstart="return start(event)"
ondragend="return end(event)">3
</div>
</div>
<div id="target2"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
<div id="target3"
ondragenter="return enter(event)"
ondragover="return over(event)"
ondrop="return drop(event)">
</div>
</body>
</html>
```