ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA BIBLIOTEKE ZA GENERISANJE I MODIFIKOVANJE MREŽNE TOPOLOGIJE UPOTREBOM SVG ELEMENTA

Master rad

Kandidat:

Nevena Pavlica 2013/3332

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2015

SADRŽAJ

1. UVOD

S obzirom da je tema ovog rada crtanje mrežne topologije, najpre ćemo se pozabaviti definisanjem samog pojma mrežne topologije.

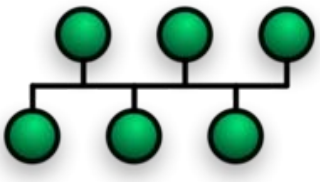
Pod pojmom mrežne topologije se podrazumevaju načini, vrste i strukture povezivanja računarskih mrežnih elemenata. Mrežnom topologijom se opisuje raspored i veze između mrežnih čvorova, a sama struktura topologije u velikoj meri određuje putanju podataka kroz mrežu. Mrežna topologija se deli na fizičku i logičku.

Fizička topologija se dalje deli na: point-to-point, magistrala, zvezda, drvo, prsten i meš.

- **Point-to-point**: ovakav način povezivanja elemenata je najjednostavniji jer se ova topologija sastoji samo iz dva čvora koji svoju komunikaciju ostvaruju direktno preko linka koji ih povezuje.

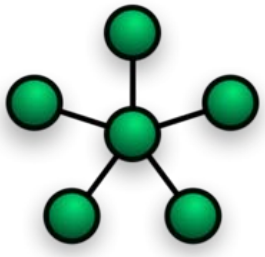
Sl.1.1. - Prikaz mrežne topologije poin-to-point

- **Magistrala**: u okviru ovakve vrste topologije postoji centralni link (magistrala) na koji su povezani ostali čvorovi. Sva komunikacija između čvorova se odvija upravo preko tog centralnog linka.



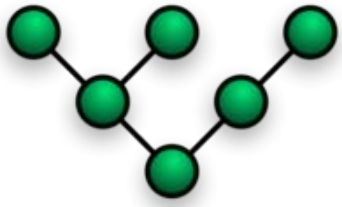
Sl.1.2. - Prikaz mrežne topologije magistrala

- **Zvezda**: ovo je ujedno i najčešći oblik povezivanja elemenata u računarskoj mreži. Ova mrežna topologija se sastoji od centralnog čvora na koji su povezani svi ostali čvorovi. Sva komunikacija u mreži ide preko centralnog čvora.



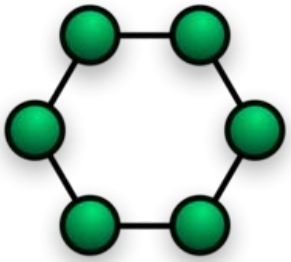
Sl.1.3. - Prikaz mrežne topologije zvezda

- **Drvo**: ovu topologiju čini koren čvor koji zauzima najvišu poziciju u hijerarhijskom lancu. Što je čvor na višoj hijerarhijskoj poziciji unutar stabla više saobraćaja ide preko njega.



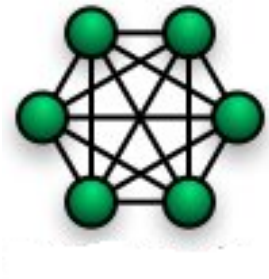
Sl.1.4. - Prikaz mrežne topologije drvo

- **Prsten**: ovu topologiju čine kružno raspoređeni čvorovi, praveći tako oblik prstena pa otud i naziv ove topologije. Podaci kruže od jednog do drugog čvora i to najčešće samo u jednom smeru. Postoji, naravno, i dvostruki prstenovi sa dve veze između čvorova gde se druga veza koristi u slučaju prekida prve veze.



Sl.1.5. - Prikaz mrežne topologije prsten

- **Meš**: U okviru ovakvog načina povezivanja čvorova moguće je ostvariti vezu sa više ili sa svim čvorovima. U pitanju je *partial* odnosno *full mesh* (svi čvorovi su međusobno povezani) topologija. Na slici 1.6 prikazana je *full mesh* topologija.



Sl.1.6. - Prikaz mrežne topologije mesh

Logička topologija se bavi načinima pristupa medijumu za slanje podataka i nije predmet ove teze. Mi ćemo se u ovom radu pozabaviti crtanjem fizičke mrežne topologije upotrebom programskog jezika HTML5 i njegovog SVG elementa.

U drugom poglavlju ćemo se osvrnuti na početak nastanka HTML programskog jezika, a biće opisan i razvoj HTML standarda. Navešćemo sve sličnosti i razlike između poslednje verzije tj. HTML5 i njegovih prethodnika. Zatim, u trećem poglavlju detaljnije ćemo se posvetiti SVG elementu kao inovaciji na polju grafike i dizajna u najnovijoj verziji HTML programskog jezika. Kroz primere ćemo prikazati neke od mogućnosti ovog elementa. Potom, u četvrtom poglavlju ćemo detaljno opisati javascript biblioteku za dinamičko kreiranje i modifikovanje mrežne topologije upotrebom upravo svg elementa. Funkcije koje su napisane omogućavaju crtanje, dodavanje, brisanje i ažuriranje čvorova, linkova i tokova u mrežnoj topologiji. Biblioteka je tako napisana da se vrlo lako mogu dodavati i oduzimati parametri koji određuju svaki od pomenutih elemenata. I na kraju, u petom poglavlju je dat primer Dijkstra algoritma realizovanog upotrebom ralizovane biblioteke za iscrtavanje mrežne topologije primenom SVG elemenata.

2.HTML

HTML (*Hyper Text Marcup Language*), pojam koji se odnosi na programski jezik koji je ujedno i osnova svake veb stranice. Korišćenjem HTML programskog jezika korisniku, tj. programeru se daje mogućnost izbora formatiranja stranice, obrade podataka, bilo da se radi o slici, video zapisu ili pak tekstu. Njime se mogu posebno odvojiti elementi kao što su paragrafi, naslovi, podnaslovi, a potom umetanjem svakog od pomenutih elemenata među tag-ove <style></style> formatirati ih u željeno stanje i oblik.

Zbog pomenutih mogućnosti, HTML spada u grupu statičkih programskih jezika jer ne postoji veza sa bazom podataka već se odnosi isključivo na izgled željene veb stranice.

Zanimljivo je napomenuti da se u okviru veb stranice nalaze takozvani „meta“ podaci koji u sebi sadrže podatke o autoru i kratak opis dokumenta, ali ovi podaci su jasno odvojeni od samog sadržaja stanice, tj. nisu vidljivi sve dok korisnik ne aktivira opciju kojom oni postaju vidljivi.

Konkretno, u ovom radu bavićemo se najviše HTML5 programskim jezikom, ali pre toga objasnićemo kako se razvijao HTML programski jezik i zašto se javila potreba za HTML5.

Sl.2 - Logo HTML5 programskog jezika

2.1. ISTORIJA I STANDARDIZACIJA

HTML je nastao uprošćavanjem SGML (*Standard Generalized Marcup Language*) standarda čija je namena bila izgled teksta koji se objavljuje na veb stranicama, ali naravno njegove mogućnosti su bile poprilično ograničene. Paralelno sa razvojem veb-a razvijala se potreba za većim mogućnostima tog programskog jezika. I tako 1991. godine polako dolazimo do HTML programskog jezika. Tačnije, 1980. godine Tim Berners Lee, zaposlen u CERN-u predložio je sistem pod nazivom ENQUIRE za slanje dokumenata, a onda 1989. godine napisao je dokument koji predlaže korišćenje hipertekst sistema na Interenetu. Njegovi raniji projekti nisu naišli na podršku da bi prvi objavljeni dokument bio “HTML Tags” pred kraj 1991. godine. Ovaj dokument

su činili elementi tj. tagovi koji su predstavljali upravo programski jezik HTML, njegov izgled. Promene koje su usledile su se odnosile na formatiranje tabela, slika i teksta. 1994. Tim Berner Lee na MIT institutu osniva W3C (*World Wide Web Consortium*) koji je usvojen kao standard prema kojem će sve veb stranice u budućnosti raditi na isti način.

IETF 1994.godine formira HTML radnu grupu koja godinu dana kasnije objavljuje HTML2.0 standard koji ujedno predstavlja bazu za svu buduću nadogradnju. Naredne godine IETF zatvara gore pomenutu radnu grupu zbog sukoba inetresa, tako da W3C samostalno objavljuje naredni standard HTML 3.2 1997. godine. Manje od godinu dana je W3C trebalo da objavi naredni standard, HTML 4.0 koji je nudio tri različite varijante: strogu, tranzicionu i frameset. Podela je nastala u zavisnosti od primene određenih elemenata. HTML 4.01 iz 1999. godine je takođe imao ove tri varijante. Poslednji standard je trenutno HTML 5. Rad na njemu počinje 2004. godine kada se W3C-u priključuje WHATWG (*Web Hypertext Application Technology working Group*). Ove dve organizacije su zajedno radile na ovom projektu sve do 2012. godine kada odlučuju da se razdvoje. WHATWG nastavlja da radi na HTML kao "živom standardu" tj. kao na standardu koji se konstanto ažurira i poboljšava. Ipak, ove dva tima sarađuju i imaju isti cilj, a to je unapređenje ovog standarda kao i podrška za nove multimedijalne sadržaje. U pitanju je cross-platforma što znači da nije važno da li se dokument otvara pomoću pametnih telefona, tableta ili laptopa. Cross-platforma je samo jedna od inovacija. Pored nje postoji još nekoliko ideja kojima se vode rukovodioci ovog projekta, kao što su npr. bolja obrada greške, veće oslanjanje na CSS i Javascript, kao i manja upotreba pluginova (Flash, Java, Silverlight) .

Istorijat HTML standarda:
1. HTML 2.0 (1995.)
2. HTML 3.2 (14.1.1997.)
3. HTML 4.0 (18.12.1997.)
4. HTML 4.01 (24.12.1999.)
5.HTML 5 (10.2014.)

2.2. POREĐENJE HTML 5 SINTAKSE SA STARIJOM VERZIJOM

Sintaksa HTML5 jezika se više ne zasniva na SGML uprkos velikoj sličnosti, već postoji veća sličnost sa HTML4.0 i XHTML 1. Na primeru koji sledi ćemo najbolje uočiti sličnosti i razlike ove dve verzije.

HTML 4.0

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<title>Naslov!</title></head>
<body>
<p>Tekst.</p>
 </body>
</html>
```

```
<!doctype html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Naslov!</title>
 </head>
 <body>
 <p>Tekst.</p>
 </body>
</html>
```

Kada pogledamo ova dva dokumenta, već u prvom redu možemo uočiti razliku. Isto zapažanje možemo postići i kada uporedimo HTML 5 sa XHTML 1 verzijom. Sintaksa nove verzije je jednostavnija. U ranijim verzijama HTML programskog jezika postojalo je više načina za definisanje doctype-a, a sada postoji samo jedan način, <!DOCTYPE html>. Druga razlika u sintaksi je što veb programeri imaju tri mogućnosti kako da definišu kodiranje znakova:

- Na transporton nivou korišćenjem HTTP Content type zaglavlja
- Korišćenjem BOM (*Byte Order Mark*) karaktera na početku dokumenta ili pak
- Korišćenjem meta elementa <meta charset="UTF-8">

"text/html" je upravo Content Type i pomoću njega se određuje kako i u kom obliku će se očitavati sadržaj i koji mu je izvor. Na primer, slike imaju svoj Content Type, image/jpeg, CSS i JavaScript imaju svoje. Moglo bi se reći da se Web zasniva na ovom kodu.

Poslednjim kodom se zamenjuje i znatno uprošćava stariji kod:

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">, ali je njegova upotreba i dalje dozvoljena.

HTML5 ne podržavaju svi brauzeri, a HTML4 podržavaju svi. Što se tiče elemenata, neki su dodati, dok su neki izbačeni iz upotrebe.

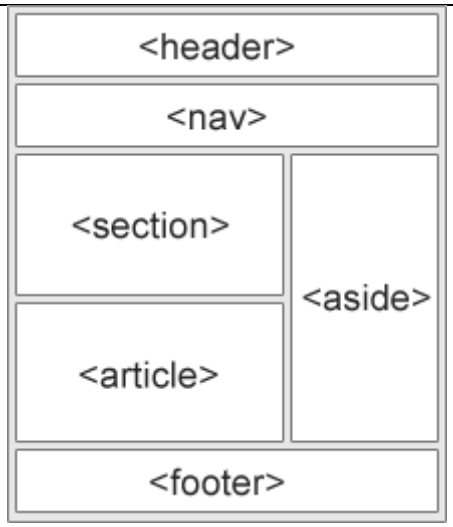
2.2.1. Novi strukturni elementi

Tabela 2.2.1. Tabela novih elemenata u HTML 5

Element	**Definicija**
<article>	Definiše članak
<aside>	Definiše sadržaj
<bdi>	Omogućava da se teo teksta posebno obradi
<command>	Komandno dugme
<details>	Def. detalje koje korisnik može da vidi/ne
<summary>	Vidljiv naslov za details tag
<figure>	Definiše dijagrame, ilustracije..
<figcaption>	Naslov za *figure* tag
<footer>	Definiše footer
<header>	Definiše header
<mark>	Definiše označeni tekst
<meter>	Definiše skalarno merenje
<nav>	Navigacioni linkovi
<progres>	Napredak nekog zadatka
<section>	Definiše sekciju
<time>	Definiše datum I vreme
<template>	Označava deo teksta koji može da se umnoži
<wbr>	Definiše prekid linije

Iz tabele 2.2.1 ćemo posebno izdvojiti *semantičke elemente*. To su elementi koji kako programerima tako i brauzeru svojim imenom jasno stavljaju do znanja svoju funkciju.

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



Sl.2.2.1. – Izgled veb stranice.

Ranije su programeri koristili, u okviru HTML 4 programskog jezika, atribute za stilizovanje veb stranica. Samo neki od njih su: footer, header, top, bottom, navigation…, ali to je stvaralo određene problem prilikom identifikovanja sadržaja stranice. Zato se uvode elementi poput <header>, <footer>, <nav> i stvari postaju nešto lakše.

Da bi lakše uočili pojednostavljenost nove sintakse, navešćemo primer za tag <nav> i tag <header>

HTML 4

```
<div id=”header”>
<div id=”navigation”>
<ul>
<li><a href=”#”>A</a></li>
<li><a href=”#”>B</a></li>
</ul>
</div>
</div>
```

HTML 5

```
<header>
<nav>
<ul>
<li><a href=”#”>A</a></li>
<li><a href=”#”>B</a></li>
</ul>
</nav>
</header>
```

Pored gore navedenih elemenata, u HTML 5 susrećemo se sa još nekim novim elementima.

2.2.2. Medijski elementi

U HTML4 smo morali dosta da se oslanjamo na plugin-ove i neke druge softvere u želji da ostvarimo interaktivnu funkciju na veb stranicama, kada govorimo o videu, igricama, muzici.., neretko smo tada nailazili na problem nekompatabilnosti sa nekim brauzerima. U HTML5 susrećemo se po prvi put sa elementima koji nam umnogome olakšavaju ovaj problem. Podrška za audio i video sadržaj ugrađena u sam HTML čime je pojednostavljeno korišćenje multimedijalnih sadržaja na samom sajtu.

Tabela 2.2.2 Tabela novih medijskih elemenata.

Element	Definicija
<audio>	Definiše zvučni sadržaj
<video>	Definiše video
<source>	Definiše vezu ka datotekama za audio i video
<emdeb>	Definiše container
<track>	Definiše tekstualne trake za audio I video

<source> elementom dobijamo mogućnost da imamo audio i video fajlove na različitim lokacijama kao i drugačijeg formata, a pretraživač tada može da otvori onaj format koji podržava. Sve do nastanka HTML 5 nije postojao standard kojim se omogućava reprodukcija audio sadržaja na veb stranici. Dosad se to obavljalo pomoću plugin-ova (kao na primer pomoću Adobe Flash-a), ali sada sa novim medijskim elementom, <audio>, je to omogućeno. Potrebno je samo definisati audio tag i navesti njegovu lokaciju. Funkcioniše pomoću atributa *controls* kojim mu se dodaju mogućnosti poput pause/play/stop/volume. Uobičajni audio formati su MP3 - audio format od MPEG (*Moving Pictures Experts Group*), AAC - *Advanced Audio Coding*, standardni format za Iphone, Youtube, OGG. Najpoznatiji brauzeri poput Chrom-a, Internet Explorer-a, Opera-e i Safari-a ovaj element, tj. formate audio sadržaja skoro sve podržavaju. Identični slučaj je i sa elementom <video>.

2.2.3. Atributi

Atributi u HTML-u obezbeđuju informaciju o sadržaju. Mnogi elementi imaju više vrsta atributa ne bi li obezbedili više informacija za svoj sadržaj. Atributi se dodaju na početku taga, nikad na kraju. Sada kada smo naveli osnovne osobine atributa možemo se okrenuti na inovacije u novom standardu. U HTML 5 dobijamo nove atribute za elemente <form> i <input>. U tabelama koje slede ćemo se upoznati sa novim atributima kao i njihovim funkcijama.

Tabela 2.2.3. Novi atributi za <form> element

Element	Definicija
Autocomplete	uključuje ili isključuje automatsko popunjavanje polja za sve kontrole na formularu
Novalidate	uključuje ili isključuje validaciju za sve kontrole na formularu

Tabela 2.2.4. Spisak novih atributa za <input> element

Element	Definicija
Autocomplete	uključuje ili isključuje automatsko popunjavanje za datu kontrolu
Autofocus	postavlja kursor za unos teksta u polje sa ovim atributom; može ga imati samo jedan element na stranici
Form, formaction, formmethod, formnovalidate	omogućuju da se kontroli dodele svojstva formulara, bez potrebe da se sam elementi nalazi u okviru formulara; ovi atributi imaju identično ponašanje kao atributi istog imena na <form> elementu.
Height/width	definišu dimenzije kontrole
List	definiše vezu na <datalist> element iz kojeg će uzimati vrednosti
Min/max	definiše najmanju i najveću vrednost za kontrole tipa "number" i "range"
Pattern	regularni izlaz po kojem se vrši provera vrednosti polja
Placeholder	predstavlja tekst koji se nalazi u polju do unosa, uglavnom predstavljen sivom bojom, koji se najčešće koristi kao vizuelni ključ o svrsi polja ili tipu/formatu sadržaja
Required	označava polje koje obavezno mora imati vrednost
Step	definiše korak povećanja vrednosti za kontrole tipa "number" i "range"

2.2.4. Tipovi kontrola

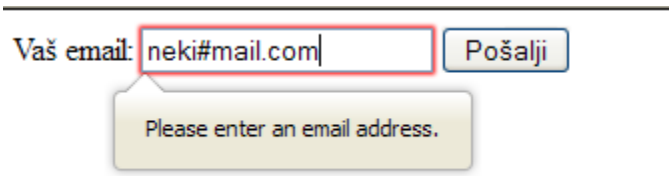
Do uvođenja standard HTML 5 bili smo ograničeni prilikom unosa podataka, tj.oslanjali smo se na neke vrste poput: text, password, radio, button, submit, checkbox. Pored navedenih, susrećemo se sa još nekoliko elemenata koji olakšavaju unos podataka.

Tabela 2.2.5-Spisak novih kontrola

Element	Definicija
Color	Boja u heksadecimalnom zapisu
Date, time, month, week	Različiti formati datuma
Email	Email adresa
Number	Broj
Range	Opseg brojnih vrednosti
Search	Pretraga
Tel	Telefonski broj
Url	Web adresa

Uzmimo na primer email. Ova polja služe za unos i validaciju (proveru ispravnosti) email adresa. Sledeći kod prikazuje primer jednog polja za unos email adrese:

```
<form action="" method="get">
<label for="email">Vaš email:</label>
<input id="email" type="email" name="email" />
<button type="submit"> Pošalji </button>
</form>
```



Sl.2.2.4.1. - Prikaz polja za unos mejl adrese

U slučaju pogrešno otkucane email adrese dobićemo iskačuću poruku koja nas o tome obaveštava.

Drugi primer na kom ćemo prikazati novitete u ovoj oblasti je color-element koji nam daje mogućnost biranja boja iz čitavog niza boja i njihovih nijansi. Poziva se jednostavnom naredbom *<input type="color">*. Ali za sad ovu mogućnost podržava samo Opera 11.

Kod kojim dobijamo taj širok spektar boja sledi:

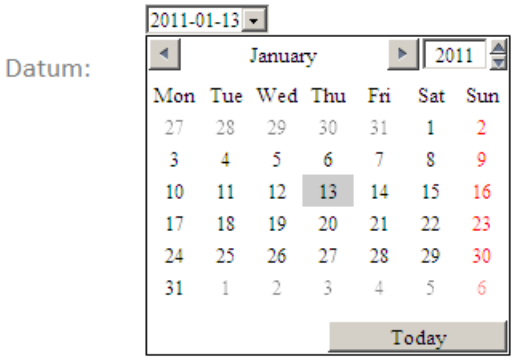
```
<label for="background-color">Odaberite boju:</label>
<input id="background-color" type="color" />
```

A klikom na opciju "other" imamo mogućnost izbora drugih boja.

Sl.2.2.4.2. - Prikaz izbora boja.

Kada god imamo potrebu za rezervacijom putovanja i nekog kulturnog događaja poput koncerta, pozorišta.., neretko se susrećemo sa kalendarom tj. opcijom odabira datuma na veb stranicama. Do sada su programeri morali da se oslanjaju na javascript date picker kod, a uz novine koje nam donosi HTML 5 dobijamo taj kalendar uz pomoć vrlo jednostavnog koda koji sledi:

```
<label for="datum">Datum : </label>
<input id="datum" type="date" value="2011-01-13"/>
```



Sl.2.2.4.3. - Prikaz izgleda kalendara

2.2.5. Grafika

Web je oduvek bio vizuelni medijum, ali je u isto vreme bio i najograničeniji. Što se tiče grafike programeri su morali da se oslanjaju na CSS i plugin-ove za potrebe animacija, ali sada se uvedeni novi elementi koji umnogome povećavaju mogućnosti za grafiku na vebu. Ti elementi su:

- 2D Canvas
- WebGL
- SVG

2D Canvas - HTML5 Canvas elementi koriste JavaScript za crtanje raznih oblika na veb stranici. Canvas pretstavlja "platno", pravougaonik nad kojim imamo potpunu kontrolu, kontrolišemo svaki piksel. Canvas element ima nekoliko metoda za crtanje putanja, pravougaonih figura, krugova, krivi, karaktera kao i za dodavanje slika.

WebGL (*Web-based Graphics Library***)** - je biblioteka koja unapređuje sposobnosti JavaScript-a. Omogućava kompatabilnim pretraživačima prikaz interaktivnih 3D animacija. Sastavni je deo Canvas elementa koji omogućava 3D kompjutersku grafiku bez korišćenja plugin-ova, pristupa mu se preko DOM interfejsa. WebGL je zasnovan na OpenGL ES 2.0 i omogućava API za 3D grafiku.

Podržani operativni sistemi:

- Windows 7/Windows 8
- Mac OS 10.6 ili novije verzije
- Linux
- Chrome OS

SVG (Scalable *Vector Graphics*) - jezik za opisivanje 2D vektorske grafike. Koristi se za definisanje grafike na vebu koja se bazira na vektorima, definiše i grafiku u XML formatu. SVG grafika ne gubi na kvalitetu ukoliko se zumira ili promeni veličina. Svaki element i atribut se može animirati. SVG saradjuje I sa ostalim W3C standardima kao što su DOM i XSL.

Više o grafici će biti reči u sledećem poglavlju.

2.2.6. Izbačeni elementi

I na kraju ovog poglavlja ćemo se osvrnuti na elemente koji su se ranije koristili u starijim verzijama HTML programskog jezika, tačnije u HTML 4, a koji su uklonjeni u novoj verziji.

- <acronym>
- <applet>
- <basefont>
- <big>
-
- <dir>
- <font>
- <frame>
- <frameset>
- <isindex>
- <noframes>
- <strike>
- <tt>

Neki od ovih elemenata su izbačeni iz novog standarda jer se njihova prezentacija i funkcija u punoj veličini može postići upotrebom CSS-a, neki jer su unosili konfuziju, neki jer se nisu koristili uopšte, a neki jer se njihova funkcija mogla postići upotrebom nekog drugog elementa.

Do sada smo predstavili HTML programski jezik od njegovih ranih početaka, tačnije od 1991. godine pa sve do danas. Potom smo ukazali na neke od izmena u sintaksi koja na neki način olakšava već postojeću, a definitivno programerima daje veće mogućnosti prilikom izrade veb sajtova. Samo nakratko smo spomenuli mogućnosti koje nam HTML5 pruža u oblasti grafike. U poglavlju koje sledi posvetićemo se detaljnije grafici, a najveću pažnju ćemo posvetiti SVG elementu kao glavnom predmetu ovog rada.

3.SVG

SVG (*Scalable Vector Graphics*) predstavlja element za 2D grafiku i grafičke aplikacije. Najčešće se koristi za crtanje dijagrama i dvodimenzionalnih sistema. 14.1.2003. godine W3C ga objavljuje. SVG je posebna vrsta DOM elementa, ali ima istu sintaksu kao HTML elementi. Ovaj element ima posebne attribute koji mu omogućavaju da definiše određene oblike. Postoje tri prednosti u korišćenju SVG elementa za crtanje grafike u odnosu na korišćenju slika u formatima JPEG, PNG itd.:

- SVG ima mogućnost velike kompresije fajlova; grafika definisana u SVG je manje veličine nego njegovi ekvivalenti u gore spomenutim formatima.
- SVG grafika se prilagođava gotovo svakoj rezoluciji bez gubitka jasnoće. Tada ti oblici imaju perfektne ivice, oštre kao žilete na svim desktop kao i na mobilnim ekranima.
- I treće, mogu da se animiraju pojedine komponente SVG elementa tokom run-time-a.

Većina brauzera prikazuje SVG isto kao što prikazuje JPG, PNG, GIF elemente, ali korisnici Internet Explorer-a moraju da instaliraju *Adobe SVG Viewer* da bi bili u mogućnosti da vide SVG u brauzeru. HTML 5 omogućava ugradnju SVG elementa direktno koristeći tag <svg>…..</svg> .

Crtanje SVG grafike se može postići upotrebom DOM elementa da bi se prikazao svaki deo tog elementa ili pak korišćenjem programa za obradu fotografija da bi se dobio određeni element koji se potom kopira u HTML kod.

U narednim stranicama, prikazaćemo koje mogućnosti u oblasti grafike nam daje SVG element.

3.1. SVG KRUG

Za razliku od HTML-a, SVG pozicioniranje se ne obavlja uz pomoć komandi, poput: top/right/bottom/left, float, ili komandama za margine. Za definisanje kruga koristimo vrednosti x i y kao vrednosti udaljenosti od centra tog kruga (odnosno cx i cy). Postoji još jedan atribut za definisanje kruga. To je radius, r.

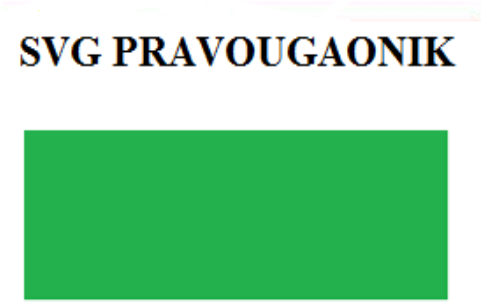
```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
<body>
<h2>SVG KRUG</h2>
<svg id="svgelem" height="200" >
<circle id="redcircle" cx="50" cy="50" r="50" fill="red" />
</svg>
</body>
</html>
```

Sl.3.1. - Prikaz crtanja kruga upotrebom SVG elementa

3.2. SVG PRAVOUGAONIK

Kod pravougaonika imamo dve vrednosti za njegovo definisanje, kao i kod kruga, s tim što je x-left, a y-top (vrednost u odnosu na gornji levi ugao). Drugi način za njegovo definisanje je uz pomoć width/ height atributa.

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
<body>
<h2>SVG PRAVOGAONIK</h2>
<svg id="svgelem" height="200" >
   <rect id="greenrect" width="300" height="100" fill="green" />
</svg>
</body>
</html>
```

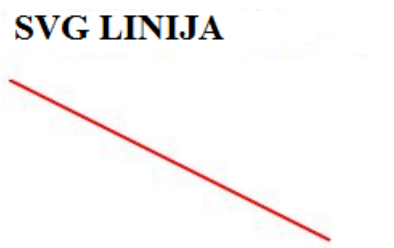


Sl.3.2. - Prikaz crtanja pravougaonika upotrebom SVG elementa.

3.3. SVG LINIJA

Da bi nacrtali liniju neophodni su atributi koji se odnose na početak i kraj te linije, a oni su: x1, y1, x2, y2.

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
<body>
<h2>SVG LINIJA</h2>
<svg id="svgelem" height="200" >
   <line x1="0" y1="0" x2="200" y2="100"
       style="stroke:red;stroke-width:2"/>
</svg>
</body>
</html>
```



Sl.3.3. - Prikaz crtanja linija upotrebom SVG elementa

3.4. SVG ELIPSA

Crtanje elipse se ne razlikuje mnogo od crtanja kruga. Rx i ry predstavljaju x i y radijuse elipse, dok su cx i cy, x i y pozicije u odnosu na centar elipse, respektivno.

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
<body>
<h2>SVG ELIPSA</h2>
<svg id="svgelem" height="200" >
   <ellipse cx="100" cy="50" rx="100" ry="50" fill="red" />
</svg>
</body>
</html>
```

Sl.3.4. - Prikaz crtanja elipse upotrebom SVG elementa

3.5. SVG TROUGAO

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
<body>
<h2>HTML5 SVG TROUGAO</h2>
<svg id="svgelem" height="200" >
   <polygon  points="20,10 300,20, 170,50" fill="red" />
</svg>
</body>
</html>
```

Sl.3.5.Prikaz crtanja trougao upotrebom SVG elementa

3.6. SVG POLYLINE

Polyline predstavlja skup ravnih linija, a sve tačke koje ih opisuju su sadržane u jednom atributu. Između tačaka su razmaci i zarezi. Svaka tačka se sastoji iz dva broja koja predstavljaju x i y koordinatu.

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />
</head>
```

```
<body>
<h2>SVG Polyline</h2>
<svg id="svgelem" height="200" >
 <polyline points="0,0 0,20 20,20 20,40 40,40 40,60" fill="red" />
</svg>
</body>
</html>
```

Sl.3.5.Prikaz crtanja polyline upotrebom SVG elementa

3.7. ANIMACIJA

Kada želimo da izvršimo animaciju elementa koristimo javascript funkciju koju pozivamo više puta. Ta funkcija menja poziciju ili dimenziju određenog oblika. Kada se funkcija pozove više puta sa kratkim intervalima između njih efekat koji postižemo je brzo menjanje položaja, odnosno dimenzije čime se izvršava animacija.

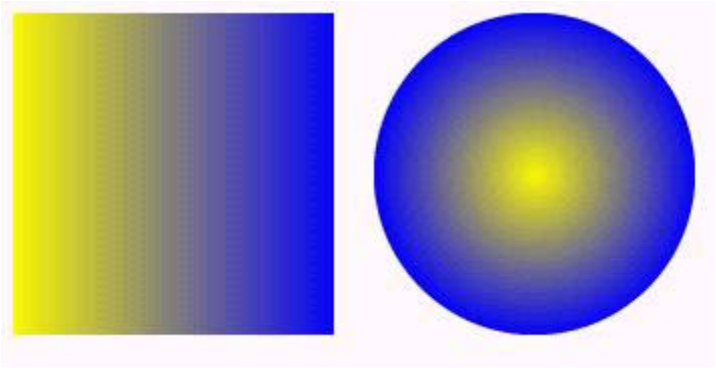
3.8. STIL

SVG elementi se mogu uređivati atributima ali naravno i korišćenjem CSS-a. Postoje nekoliko različitih kategorija kada govorimo o stilizovanju elemenata:

- ***Color***: koji se sastoji od *fill* I *stroke*. *Fill* je ekvivalent sa *background-color* u CSS-u, dok je stroke isto što i *border-color*.
- ***Gradient***: način bojenja oblika uz pomoć prelivanja boja. Razlikujemo dve vrste: linearni gradijent i radijalni gradijent. Uključuje *stopColor, stopOpacity, offset* attribute za definisanje višenamenskih gradijenata.

  stopColor:boja se menja od/do

  stopOpacity:providnost boje
- ***Dimenzija***: definiše veličinu i poziciju nekog elementa



Sl.3.8.1. - Primer gradijenta

4. DINAMIČKO KREIRANJE MREŽNE TOPOLOGIJE

Na početku ovog rada bavili smo se varijantama mrežne topologije. U okviru ovog poglavlja pozabavićemo se njenim crtanjem uz pomoć HTML5 elementa, SVG i uz pomoć JavaScript biblioteke.

Čvorovi kao i linkovi mogu se dodavati, ali i ukljanjati iz mreže. U ovom poglavlju ćemo dati funkcije kojima se to postiže.

4.1. DODAVANJE ČVORA, LINKA I TOKA

1. Čvor

Tri osnovna elementa koja čine mrežnu topologiju su čvorovi, linkovi kao i tokovi. Svaki od ovih elemenata je potrebno definisati. Prvi koji ćemo definisati uz pomoć koordinata koje opisuju njegov položaj u prozoru i kašnjenje je čvor uz pomoć funkcije ***defineNode***.

```
function defineNode(nx,ny,nz, isNodeDrawn){
        this.nx = parseInt(nx);
        this.ny = parseInt(ny);
        this.nz = parseFloat(nz);
        this.isNodeDrawn = isNodeDrawn;
        console.log("define node: ",nx,", ",ny,", ",nz,", ",isNodeDrawn);
    }
```

Promenljiva *nx* je X koordinata, a promenljiva *ny* je Y koordinata. Promenljiva *nz* označava kašnjenje datog čvora. Korisnik preko korisničkog interfejsa zadaje vrednosti ovih parametara i one se prosleđuju funkciji ***defineNode***, u kojoj se vrši konverzija argumenata koji su tipa string u format broja pomoću funkcija ***parseFloat()*** i ***parseInt()***. 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 *nx*, *ny* i *nz* se zatim postavljaju kao osobine objekta *Node*, a to su *nx, ny* i *nz*.

Da bi kreirali novi objekat tipa node koristimo naredbu:

```
var node = new defineNode;
```

Za čuvanje niza objekata tipa node predviđen je niz *nodes*:

```
var nodes = [];
```

Za kreiranje niza koji sadrži objekte node koristimo metodu *push()* tj. ***nodes.push(node);***.

Ovom metodom dodajemo element u niz, tj. novi element ide na kraj niza, a indeksiranje elemenata kreće od nule.

2.Link

Drugi element koji treba da definišemo je link. Uradićemo to na vrlo sličan način kao i čvor.

Parametri koji opisuju link su identifikacija početnog i završnog čvora, cena, kapacitet i kašnjenje linka. Funkcija kojom definišemo ovaj objekat se naziva ***defineLine*** je:

```
function defineLine(n1,n2,startNodeX,startNodeY,endNodeX,endNodeY,
kasnjenjeLinka,kapacitetLinka, isLineDrawn) {
    this.n1 = parseInt(n1);
        this.n2 = parseInt(n2);
        this.startNodeX = parseInt(startNodeX);
        this.startNodeY = parseInt(startNodeY);
        this.endNodeX = parseInt(endNodeX);
        this.endNodeY = parseInt(endNodeY);
        this.kasnjenjeLinka = parseFloat(kasnjenjeLinka);
        this.kapacitetLinka = parseFloat(kapacitetLinka);
        this.isLineDrawn = isLineDrawn;

        console.log("lines array ",lines);
    }
```

Ulazni parameteri služe da opišemo link. Parametri *startNodeX*, *startNodeY*, *endNodeX*, *endNodeY* služe za pozicioniranje linka između dva čvora, noda. Oni predstavljaju koordinate određenog čvora. Dodali smo i parameter *n1* i *n2*, koji se odnose na redne brojeve čvorova. Svi parametri, sem parametra *kapacitetLinka* i *kašnjenjeLinka*, su određeni celim pozitivnim brojevima, pa se parametar *kapacitetLinka* i *kašnjenjeLinka* konvertuje iz tipa *string* u tip *float*, a svi ostali se pomoću funkcije ***parseInt()*** konvertuju iz tipa *string* u tip *integer*. Posle konverzije, vrednosti ulaznih parametara se postavljaju kao vrednosti osobina objekta ***Line***. Još jedan parameter smo uveli, *isLineDrawn*, koji je tipa *boolean*, što znači da može imati samo dve vrednosti, odnosno može biti *true* ili *false*.

Novi objekat tipa Line se može kreirati naredbom:

```
var line = new defineLine
```

a kao što je bio slučaj i sa node-ovima tako i ovde definišeno niz *var lines = [];* za skladištenje objekata tipa defineLine. Dodavanje elemenata u niz *Lines* se vrši po istom principu kao kod nodes korišćenjem metode **push().**

lines.push(line);.

3.Tok

Poslednji od objekata koje treba da definišemo je tok. Funkcija pomoću koje ćemo definisati tok je ***defineFlow*** :

```
function defineFlow(n1,n2,startFlowX, startFlowY, endFlowX, endFlowY,
kapacitetToka, isFlowDrawn){
        this.n1 = parseInt(n1);
        this.n2 = parseInt(n2);
        this.startFlowX = parseInt(startFlowX);
        this.startFlowY = parseInt(startFlowY);
        this.endFlowX = parseInt(endFlowX);
        this.endFlowY = parseInt(endFlowY);
        this.kapacitetToka = parseFloat(kapacitetToka);
        this.isFlowDrawn = isFlowDrawn;
    }
```

Parametri koji definišu tok podataka stoje u zagradi i odnose se na poziciju dva čvora između kojih crtamo tok kao i na kapacitet toka. Takođe, upotrebili smo *isFlowDrawn* koji je tipa *boolean.*

Novi objekat tipa ***defineFlow*** možemo kreirati naredbom:

```
var flow = new defineFlow(n1,n2,startFlowX,startFlowY, endFlowX,endFlowY
,kapacitetToka, isFlowDrawn);
```

Kreiranjem više ovakvih objekata stvaramo niz: *var flows = [];*.

Ponovo ćemo koristiti medotu ***push()*** za popunjavanje niza ***flows.***

4.2. CRTANJE ČVORA, LINKA I TOKA

Za grafičku predstavu topologije mreže u ovom radu je upotrebljen HTML5 element<*svg*>. Da bi se pronašao <*svg*> element, poziva se *JavaScript* metoda ***getElementById()***.

var svgWindow = document.getElementById("mySvg");.

Uz pomoć prethodnih, gore navedenih funkcija smo definisali čvorove, linkove i tokove. Sada treba da ih nacrtamo. Prvi element koji ćemo crtati je čvor.

Deo HTML koda koji je korišćen za generisanje stranice je, a koji se odnosi na čvor je sledeći:

```
<script src="draw.js"></script>
<div class="node">
            <h2>Unesi parametre čvora:</h2>

            <p>Koordinata X:<input type="text" id="x" ></p>
            <p>Koordinata Y:<input type="text" id="y"></p>
            <p>Kašnjenje(ms):<input type="text" id="z"></p>
            <button type="button" onclick="addNode()" >dodaj čvor</button>
            <button type="button" onclick="drawNodes()" >crtaj čvor</button>
            </div>
```

Poziva se *javascript* fajl *draw.js*. Uz pomoć dve komande :

```
<button type="button" onclick="addNode()" >dodaj čvor</button>
<button type="button" onclick="drawNodes()" >crtaj čvor</button>
```

Na veb stranici pojavlju se dva dugmeta kojim se izvršavaju javascript funkcije. Da bi postigli crtanje čvorova, koristili smo dve funkcije: ***addNode()*** i ***drawNodes(),*** koje služe za dodavanje i crtanje čvorova, respektivno.

```
function addNode(){
        var nx = document.getElementById("x").value;
        var ny = document.getElementById("y").value;
        var nz = document.getElementById("z").value;
        var isNodeDrawn = false;
            // definisanje cvora i niza

            var validation = ValidateNodeParameters(nx, ny, nz);
            console.log("Validate NODE je :",validation);
            if (validation === true){
                var node = new defineNode(nx,ny, nz, isNodeDrawn);
```

```
                nodes.push(node);
                console.log("add node push je: ",node);
            } else{
                console.log("Validation NODE ERROR");
            }
    }
```

Funkcija **addNode()** uzima vrednosti parametara čvora pomoću *document.getElementById* i pomoću njih kreira novi objekat tipa defineNode, a koji se naziva node, a potom ga dodaje u niz *nodes* pomoću metode *push()*. Kao što je već rečeno, *isNodeDrawn* može imati dve vrednosti, samo *true* ili *false*. Postavljamo je na vrednost *false* pomoću

var isNodeDrawn = false;

zato što čvor nije još uvek nacrtan. Kada funkcija izvrši svoj zadatak, ta vrednost postaje *true*. Zatim, uvodimo novu vrednost *validation* koja vraća vrednost funkcije ***ValidateNodeParametars().*** Ova funkcija vraća proveru da li su unete vrednosti koje se uzimaju iz html unosa tačne tj. verifikovane (validate.js) u slučaju da jesu, node se definiše preko *defineNode* i snima u niz *nodes*. Inače, ako vrednosti nisu tačne tj. ako vrati *false*, neće se definisati node i uneti u niz jer nema svrhe unositi netačne vrednosti.

```
function drawNode(node,i) {
    var svg = document.getElementById("mySvg");
        var element = document.createElementNS(svgns, "circle");

        element.setAttribute("cx", node.nx);
        element.setAttribute("cy", node.ny);
        element.setAttribute("r",  20);
        element.setAttribute("stroke", "black");
        element.setAttribute("stroke-width", 2);
        element.setAttribute("fill", "lightgreen");
        element.setAttribute("class","cvor");

        //broj noda
        var text = document.createElementNS(svgns, "text");
        text.setAttribute("x", node.nx + 25);
        text.setAttribute("y", node.ny + 25);
        text.setAttribute("fill", "green");
        text.setAttribute("class","text");
        text.textContent = i;

        svg.appendChild(text);
        svg.appendChild(element);

}

function drawNodes() {
    for (i=0; i<nodes.length; i++) {
        var node = nodes[i];

                //provera da li je node vec nacrtan

              if (node.isNodeDrawn === false){
                 node.isNodeDrawn = true;
                 drawNode(node, i+1);
```

```
                console.log("DRAW NODES [",i,"]: // Node nije nacrtan ",node);
              }else{
                console.log("DRAW NODES [",i,"] // Node je vec nacrtan", node);
               }

    }
}
```

Koristi se funkcija ***drawNode()*** koja je deo biblioteke napisane u ovom radu, i kojoj je potrebno proslediti referencu na *svg* element, odnosno *document.getElementById("mySvg")*, njegov kontekst, kao i niz čvorova koji treba da se iscrtaju u elementu *svg*, kao deo mrežne topologije.

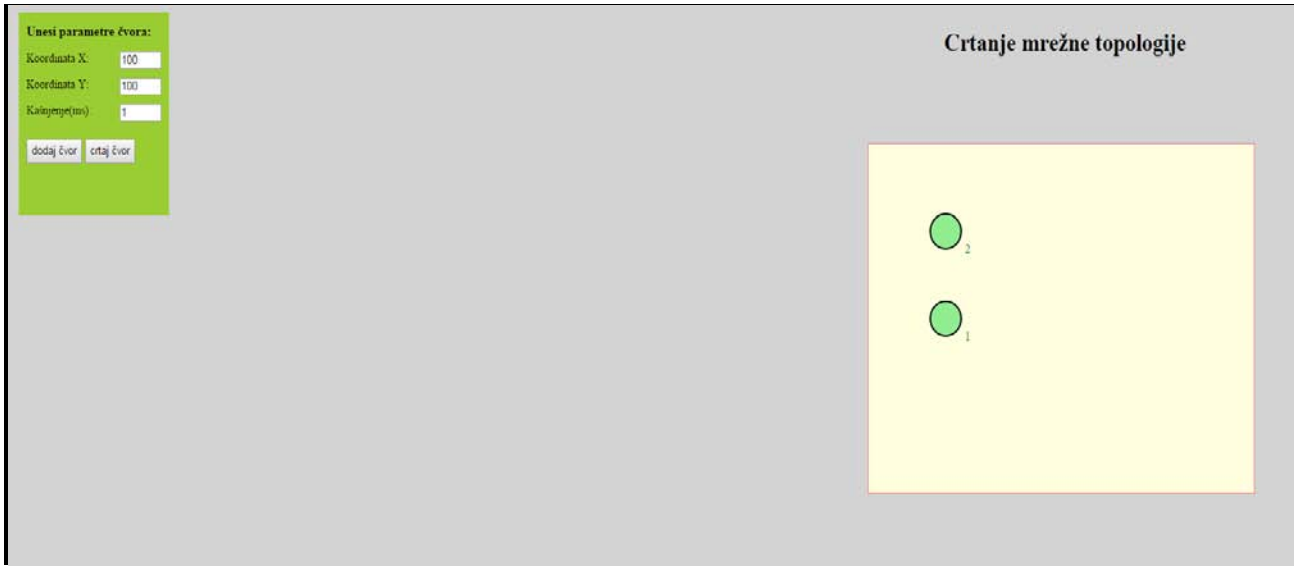
Uz pomoć *javascript* metode, *document.createElementNS(svgns, "circle",* kreiramo element krug.

A uz pomoć *element.setAttribute* određujemo dizajn kruga, tj. čvora. Na isti način dodajemo tekst u ovaj dokument metodom *var text = document.createElementNS(svgns, "text");* i uz *text.setAttribute* ga stilizujemo *svg.appendChild(text);*

Funkcija kojom proveravamo da li je čvor nacrtan ili nije je ***drawNodes().*** For petljom funkcija prolazi kroz sve elementa niza *nodes* čija je dužina određena javascript medotod *length*. U svakom prolasku kroz for petlju definiše se promenljiva *node* koja predstavlja trenutni element niza *nodes*.

Uz pomoć sledećih naredbi proveravamo da li je čvor već nacrtan. Ukoliko on već postoji, pojavljuje nam se obaveštenje da je node već nacrtan, i tada se ne poziva funkcija ***drawNode*** u suprotnom node nije nacrtan i poziva se funkcija ***drawNode***.

```
if (node.isNodeDrawn === false){
                node.isNodeDrawn = true;
                drawNode(node, i+1);
                console.log("DRAW NODES [",i,"]: // Node nije nacrtan ",node);
              }else{
                console.log("DRAW NODES [",i,"] // Node je vec nacrtan", node);
               }
```



Sl.4.3.1. - Prikaz stranice za crtanje čvorova

Sledeći korak ka izgradnji mrežne topologije je crtanje linkova koje predstavljamo uz pomoć elementa *svg* kao linije koje spajaju dva čvora, tj. kruga. U prethodnom poglavlju je bilo reči o parametrima koji ga karakterišu (koordinate dva čvora, kapacitet, cena i kašnjenje) i koje korisnik unosi preko korisničkog interfejsa.

HTML kod koji je korišćen za generisanje stranice i koji se odnosi na link je sledeći:

```
<script src="draw.js"></script>
<div class="link">
            <h2>Unesi parametre linka:</h2>

            <p>Pocetni čvor :<input type="text" id="n1"></p>
            <p>Krajnji čvor :<input type="text" id="n2"></p>
            <p>Cena :<input type="text" id="cenaLinka"></p>
            <p>Kašnjenje(ms):<input type="text" id="kasnjenjeLinka"></p>
            <p>Kapacitet(kb/s):<input type="text" id="kapacitetLinka"></p>
            <button type="button" onclick="addLine()">dodaj link</button>
            <button id="drawLine" type="button" onclick="drawLines()">crtaj
link</button>
            </div>
```

Poziva se *javascript* fajl ***draw.js***. kao što je bilo reči kod čvorova, isto i kod crtanja linkova koristimo metodu :

```
 <button type="button" onclick="addLine()">dodaj link</button>
<button id="drawLine" type="button" onclick="drawLines()">crtaj link</button>
```

kojom stvaramo dva dugmeta čijim klikom izvršavamo javascript funkcije. Funkcije o kojima pričamo su ***addLine()*** I ***drawLines().*** One se odnose na dodavanje linka i njegovo grafičko predstavljanje, respektivno.

```
function addLine() {

    var n1 = document.getElementById("n1").value;
    var n2 = document.getElementById("n2").value;
        var kasnjenjeLinka = document.getElementById("kasnjenjeLinka").value;
        var kapacitetLinka = document.getElementById("kapacitetLinka").value;
        var isLineDrawn = false;
        var validation =  ValidateLineParameters(n1,n2,kasnjenjeLinka,
kapacitetLinka);
        console.log("Validate line je :",validation);

        if(validation === true){
            //nadji startni cvor u listi cvorova

            var startNode = nodes[n1-1];
            var startNodeX = startNode.nx;
            var startNodeY = startNode.ny;
            console.log("nx je",n1);

            //nadji kranji cvor u listi cvorova
            var endNode = nodes[n2-1];
            var endNodeX = endNode.nx;
            var endNodeY = endNode.ny;
```

```
            var line = new
defineLine(n1,n2,startNodeX,startNodeY,endNodeX,endNodeY,kasnjenjeLinka,
kapacitetLinka,isLineDrawn);

                lines.push(line);
                console.log("Add push line je :",line);
            }else{
                console.log("Validation LINE ERROR");
            }
}
```

Funkcija ***addLine()*** uzima vrednosti parametara čvora pomoću *document.getElementById* i pomoću njih kreira novi objekat tipa defineLine, a koji se naziva *line*, a potom ga dodaje u niz *lines[]* pomoću metode *push()*. Zatim, komandama :

```
 var startNode = nodes[n1-1];
 var startNodeX = startNode.nx;
 var startNodeY = startNode.ny;
 console.log("nx je",n1);
```

tražimo početni čvor. A uz pomoć sledećih komandi tražimo krajnji čvor.

```
var endNode = nodes[n2-1];
var endNodeX = endNode.nx;
var endNodeY = endNode.ny;
var line = new
defineLine(n1,n2,startNodeX,startNodeY,endNodeX,endNodeY,kasnjenjeLinka,
kapacitetLinka,isLineDrawn);
```

Uvodimo ponovo funkciju *ValidateLineParametars* iz istih razloga kao i kod čvorova. Funkcije koja crtaju link i proveravaju da li već postoje su ***drawLine()*** i ***drawLines().***

```
function drawLine(line) {
    var svg=document.getElementById("mySvg");
    var element = document.createElementNS(svgns, "line");


    //set line attributes
        console.log("startnodex je",line);
    element.setAttribute("x1", line.startNodeX+2);
    element.setAttribute("y1", line.startNodeY+2);
    element.setAttribute("x2", line.endNodeX+2);
    element.setAttribute("y2", line.endNodeY+2);
    element.setAttribute("stroke", "black");
    element.setAttribute("stroke-width", "2");
    element.setAttribute("class","link");

    svg.appendChild(element);
    }
```

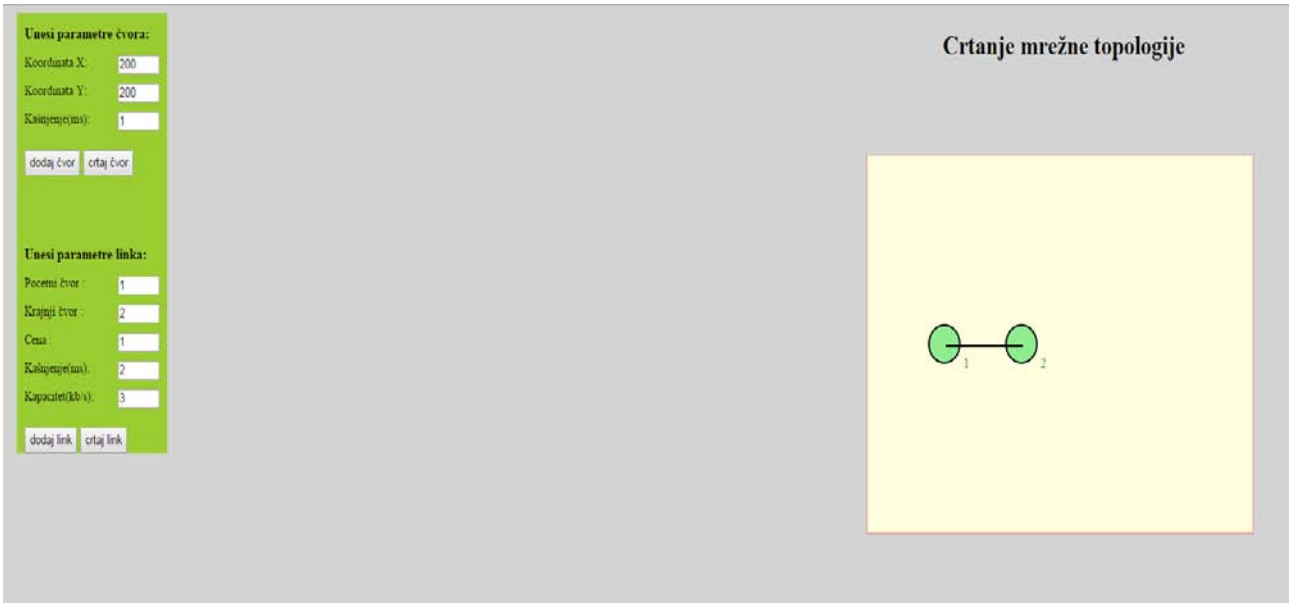
Funkcija ***drawLine()*** je deo biblioteke napisane u ovom radu, i potrebno joj je proslediti referencu na *svg* element, odnosno *document.getElementById("mySvg")*, njegov kontekst, kao i niz linkova koji treba da se iscrtaju u elementu *svg*, kao deo mrežne topologije.

Uz pomoć *javascript* metode, *document.createElementNS(svgns, "line",* kreiramo element liniju. A uz pomoć *element.setAttribute* određujemo dizajn linije, tj linka. Linija može biti različite debljine kao i boje. U ovom radu link je predstavljen crnom bojom.

```
function drawLines() {
    for(i=0; i<lines.length; i++) {
        var line = lines[i];

                //provera da li je linija vec nacrtana

                if(line.isLineDrawn === false){
                    line.isLineDrawn = true;
                    drawLine(line);
                    console.log("DRAW LINES[",i,"] Line nije nacrtan ",line);
                }else{
                    console.log("DRAW LINES [",i,"] Line je nacrtan",line);
                }

    }
}
```

Funkcija koja crta linkove je ***drawLines().*** For petlja prolazi kroz svaki element. Funkcija svake iteracije for petlje je da iscrtava po jedan link. U svakom prolazu se definiše lokalna promenljiva u ciklusu, *line*, kojoj se dodeljuje vrednost objekta niza *Lines* sa indeksom *i*. I potom se proverava da li je link nacrtan ili već postoji. Ukoliko link postoji dobijamo obaveštenje "Line je nacrtan", a u suprotom "Line nije nacrtan".

```
if(line.isLineDrawn === false){
                    line.isLineDrawn = true;
                    drawLine(line);
                    console.log("DRAW LINES[",i,"] Line nije nacrtan ",line);
                }else{
                    console.log("DRAW LINES [",i,"] Line je nacrtan",line);
                }
```



Sl.4.3.2. - Prikaz stranice za unos parametara linka

I za kraj ovog poglavlja ostalo je da prikažemo funkcije korišćene za iscrtavanje tokova. HTML kod korišćen za generisanje stranice, a koji se odnosi na iscrtavanje toka je sledeći:

```
<script src="draw.js"></script>
<div class="flow">
        <h2>Unesi parametre toka:</h2>

    <p>Početni čvor:<input type="text" id="f1"></p>
    <p>Krajnji čvor:<input type="text" id="f2"></p>
    <p>Kapacitet(kb/s):<input type="text" id="kapacitetToka"></p>
    <button type="button" onclick="addFlow()">dodaj tok</button>
    <button id="drawFlow" type="button" onclick="drawFlows()">crtaj tok</button>
 </div>
```

Poziva se javascript fajl *draw.js*. korisnik unosi željene vrednosti za početni i krajnji čvor, i za kapacitet preko korisničkog interfejsa. A uz pomoć komandi :

```
<button type="button" onclick="addFlow()">dodaj tok</button>
<button id="drawFlow" type="button" onclick="drawFlows()">crtaj tok</button>
```

Korisnik klikom na dva dugmeta, dodaj tok i crtaj tok, izvršava funkcije *addFlow* i *drawFlows*, respektivno.

```
function addFlow(){
            var n1 = document.getElementById("f1").value;
            var n2 = document.getElementById("f2").value;
            var kapacitetToka = document.getElementById("kapacitetToka").value;
            var isFlowDrawn = false;
            var validation = ValidateFlowParameters(n1,n2,kapacitetToka);
            console.log("Validate flow je :",validation);

            if (validation === true){
                //nadji startni cvor u listi cvorova
                var startFlow = nodes[n1-1];
                var startFlowX = startFlow.nx;
                var startFlowY = startFlow.ny;

                //nadji kranji cvor u listi cvorova
                var endFlow = nodes[n2-1];
                var endFlowX = endFlow.nx;
                var endFlowY = endFlow.ny;

                var flow = new defineFlow(n1,n2,startFlowX,startFlowY,
endFlowX,endFlowY ,kapacitetToka, isFlowDrawn);

                flows.push(flow);
                console.log("Add push flow je :",flow);
            }else{
                console.log("Validation FLOW ERROR");
            }


    }
```

Funkcija ***addFlow()*** uzima vrednosti parametara toka pomoću *document.getElementById* i pomoću njih kreira novi objekat tipa defineFlow, a koji se naziva *flow*, a potom ga dodaje u niz *flowes[]* pomoću metode *push()*.

Brojanje elmenata u Javascriptu i HTML-u se razlikuje, jer u HTML-u prvi element u nizu je 1, a u Javascriptu je 0, pa je potrebno oduzeti 1 da bi se dobio prvi čvor u js. Uz pomoć kodova koji slede nalazimo startni i krajnji čvor u listi čvorova.

```
var startFlow = nodes[n1-1];
                var startFlowX = startFlow.nx;
                var startFlowY = startFlow.ny;

                //nadji kranji cvor u listi cvorova
                var endFlow = nodes[n2-1];
                var endFlowX = endFlow.nx;
                var endFlowY = endFlow.ny;

                var flow = new defineFlow(n1,n2,startFlowX,startFlowY,
endFlowX,endFlowY ,kapacitetToka, isFlowDrawn);
```

Funkcija ***drawFlow()*** pomoću koje iscrtavamo tok:

```
function drawFlow(flow){
        var svg=document.getElementById("mySvg");
    var element = document.createElementNS(svgns, "line");

        element.setAttribute("x1", flow.startFlowX-2);
        element.setAttribute("y1", flow.startFlowY-2);
        element.setAttribute("x2", flow.endFlowX-2);
        element.setAttribute("y2", flow.endFlowY-2);
        element.setAttribute("stroke", "red");
        element.setAttribute("stroke-width", 2);
        element.setAttribute("stroke-dasharray",5,5);
        element.setAttribute("class","tok");

        svg.appendChild(element);
    }
```

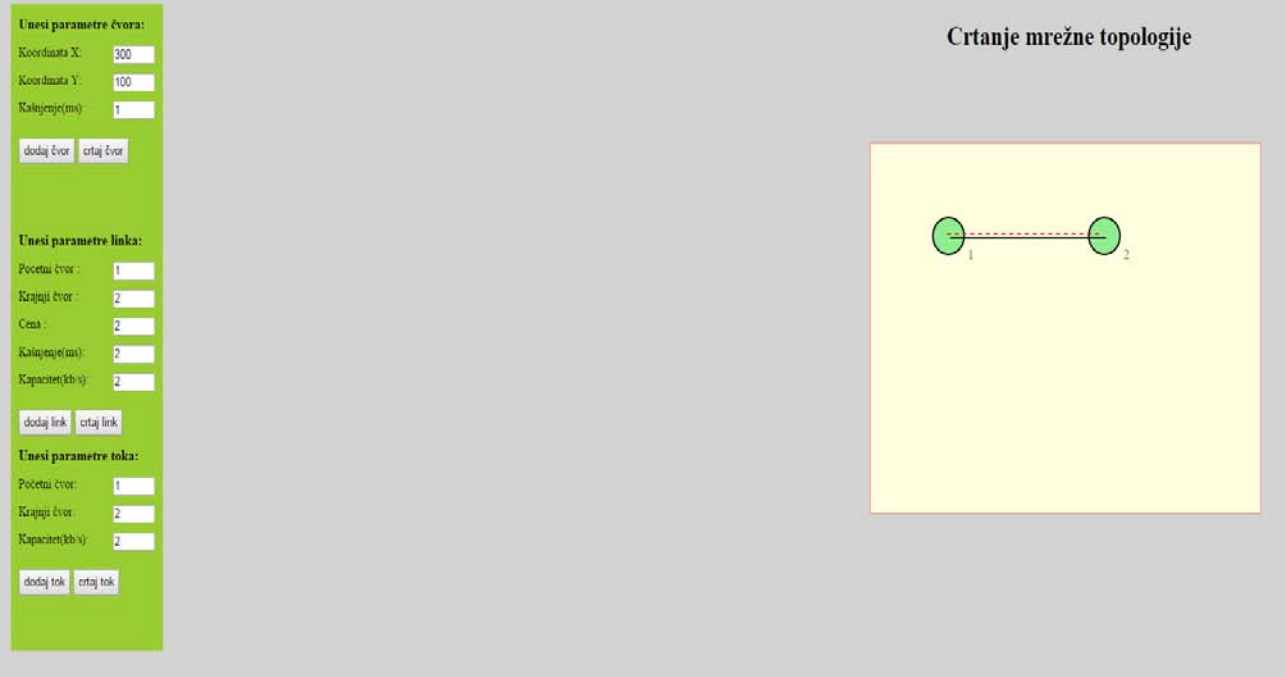
Funkcija ***drawFlow()*** je deo biblioteke napisane u ovom radu, i potrebno joj je proslediti referencu na *svg* element, odnosno *document.getElementById("mySvg")*, njegov kontekst, kao i niz linija koji treba da se iscrtaju u elementu *svg*, kao deo mrežne topologije.

Uz pomoć *javascript* metode, *document.createElementNS(svgns, "line"*, kreiramo element liniju. A uz pomoć *element.setAttribute* određujemo dizajn linije, tj. toka. U ovom radu tok je predstavljen crvenom, isprekidanom linijom.

```
function drawFlows() {
    for(i=0; i<flows.length; i++) {
                var flow = flows[i];

                //provera da li je tok vec nacrtan

        if(flow.isFlowDrawn === false){
                    flow.isFlowDrawn = true;
```

```
                drawFlow(flow);
                console.log("DRAW FLOW[",i,"] Flow je nacrtan ",flow);
            }else{
                console.log("DRAW FLOW [",i,"] Flow nije nacrtan ",flow);
            }
    }
    }
```

For petlja prolazi kroz svaki element. Funkcija svake iteracije for petlje je da iscrtava po jedanu liniju. U svakom prolazu se definiše lokalna promenljiva u ciklusu, *flow*, kojoj se dodeljuje vrednost objekta niza *Flows* sa indeksom *i*. I potom se proverava da li linija već postoji. Ukoliko linija postoji dobijamo obaveštenje "Flow je nacrtan", a u suprotom "Flow nije nacrtan".



Sl.4.3.3. - Prikaz stranice za unos parametara toka.

Ovim smo završili deo sa iscrtavanjem mrežne topologije. U daljem radu bavićemo se njihovim brisanjem, ažuriranjem i validacijom.

4.3. BRISANJE ČVORA, LINKA I TOKA

Za brisanje linka koristimo funkciju ***deleteLink().*** HTML kod koji se odnosi na ovaj deo je:

```
<h2>Brisanje linka</h2>
            <p>Početni čvor:<input type="text" id="dl1"></p>
            <p>Krajnji čvor:<input type="text" id="dl2"></p>
            <button id="deleteLink" type="button" onclick="deleteLink()">obriši
link</button>

function deleteLink(){

    var d1 = document.getElementById("dl1").value;
    var d2 = document.getElementById("dl2").value;
    var d1 = parseInt(d1);
    var d2 = parseInt(d2);
    var el = "link";
```

```
    var niz = lines;
    brisi(d1,d2,el,niz);

}
```

Kao što vidimo, parametri *dl1* i *dl2* se odnose na početni i krajnji čvor linka i oni su *integer* tipa. Kreiramo i promenljive *el* i *niz* koje se odnose na link i niz *lines*, respektivno i pozivamo funkciju ***brisi()*.**

```
function brisi(d1,d2,el,niz){
    var deleteElementInArray = false;
    var toDelete;

    for(j=0;j<niz.length;j++){
        var nizovi = niz[j];
        // za svaki niz u lines proverava da li postoji link sa pocetni i
kranjim cvorom koji je unet da se obrise

        if(((nizovi.n1 == d1) && (nizovi.n2 == d2))||((nizovi.n1 == d2)
&&(nizovi.n2 == d1))){
            //ako postoji onda je varijabla true i uzima broj tog linka iz niza
            deleteElementInArray = true;
            toDelete = j;
            console.log("to delete: ",toDelete);
        }
    }

    console.log("d1 i d2",d1,d2);
    console.log("deletelink :",deleteElementInArray);

    if(((d1<0)||(d1>nodes.length))||((d2<0)||(d2>nodes.length))||(d1 ==
d2)||(deleteElementInArray == false)){
        // ako unete vrednosti nisu validne izbacuje obavestenje
        alert("Proverite da li "+el+" koji zelite da izbrisete postoji");
        console.log("nije obrisan",el,niz.length);
    }else{
        // ako jesu brise se linija iz niza
        console.log("pre brisanja >",el,niz.length);

        niz.splice(toDelete,1);

        console.log("posle brisanja >",el,niz.length);

        BrisiElementHtml(toDelete,el);
    }
}
```

Funkcija *brisi()* ima ulazne parametre koje smo kreirali u funkciji *deleteLink()*, početni i krajnji čvor, link i niz *lines*. Promenljiva *deleteElementInArray* se postavlja na vrednost *false*. Pomoću ove promenljive i uz pomoć for petlje proverava se da li stvarno postoji link koji je zadat preko korisničkog interfejsa sa početnim i krajnjim čvorom kao link koji treba da se obriše. Ako postoji onda se smešta u promenljivu *toDelete*, a varijabla uzima vrednost *true* i uzima broj tog linka iz niza. Nakon toga se validiraju uneti podaci. Ako unete vrednosti nisu validne, tj. korisnik za čvorove uneo brojeve koji su manji od 0 ili koji su veći od dužine niza *nodes* ili pak link između njih ne postoji izbacuje obaveštenje “Proverite da li "+el+" koji želite da izbrisete postoji”, a ako jesu briše se linija iz niza. Pošto brisanje linka podrazumeva uklanjanje elementa niza sa određenim

indeksom, koristimo metodu *splice()*. Prvi parameter se odnosi na poziciju elementa koji treba ukloniti, a drugi parameter na droj elemenata koje hoćemo da obrišemo. Na ovaj način je dobijen niz *niz*, sa istim redosledom elemenata, samo bez linka koji je korisnik zadao da treba da bude izbrisan. Poziva se funkcija ***BrisiElementHtml(toDelete,el)*** kojom brišemo element iz HTML-a**.** Parametri ove funkcije su *toDelete* koji se odnosi na redni broj elementa iz niza koji je obrisan, a *el* je tok ili link koji se briše. Kreiramo promenljivu *c* koja se odnosi na sve elemente koji su nacrtani u svg-u koristeći *childNodes*. Kriramo varijablu *n1* koju postavljamo na vrednost 0. Zatim, for petljom proveravamo koliko ima nacrtanih elemenata. U promenljivu *s* stavljamo svaki svg element. Ako je *nodeType=1* to znači da je svg element. Postavljamo da je *S1=el*, što znači da može imati dve vrednosti, ili je tok ili je link. *var s2 = s.attributes["class"].value*, ovom komandom se uzima vrednost svg klase koja može biti čvor, tok ili link, ako je *s1=s2* tj. ako je link=link ili ako je tok=tok sledeća if petlja proverava komandom *n1=toDelete* da li je redni broj tog html elementa isti kao i onog što je obrisan iz niza. Ako jeste *removeChild* briše taj element iz html-a.

```
function BrisiElementHtml(toDelete,el){

    var c = svgWindow.childNodes;
        console.log("child node",c);
        console.log("duzina noda je:",c.length);
        var nl = 0;

         // proverava koliko ima elemenata nacrtanih
        for(i=0;i<c.length;i++){
            var s = svgWindow.childNodes[i];

            var s1 = el;
            console.log("Tip noda je ", s.nodeType);
           // proverava da li je tip noda svg element, jeste ako je 1
            if(s.nodeType === 1){
                var s2 = s.attributes["class"].value;

                 console.log("Klasa HTML elementa
",s.attributes["class"].value);
                if( s2 === s1 ){
                // ukoliko ima nacrtanih link,tokova
                console.log("Node ",i," jeste link ili tok");

                console.log(nl,"nl  toDelete",toDelete);

                    if(nl === toDelete ){
                        // ako je to element koji je obrisan iz niza, izbrisi i
iz html-a
                        svgWindow.removeChild(s);
                        console.log("element je izbrisan.");
                    }
                // brojac svg elementa
                nl = nl+1;
                }
}
        }}
```

Pored brisanja linka možemo i čvor da obrišemo pomoću funkcije ***deleteNode(),*** dok je html kod koji je korišćen za izradu ovog dela stranice:

```
    <h2>Brisanje čvora</h2>
     <p>Krajnji čvor:<input type="text" id="dn"></p>
     <button id="deleteNode" type="button" onclick="deleteNode()">obriši
čvor</button>

function deleteNode(){

    var dn = document.getElementById("dn").value;
    var dn = parseInt(dn);
    var validation;

    var validation = ValidateDeleteNodeParameters(dn);

    if(validation === true){
        // ako je verifikaacija cvora uspesna brisi cvor
        brisiCvor(dn);
    }else{
        console.log("validation delete node ERROR!");
    }
}
```

U funkciji formiramo promenljivu *dn* koja se odnosi na krajnji čvor koji hoćemo da obrišemo. Poziva se funkcija *ValidateDeleteNodeParametars()* koja ima ulazni parametar *dn,* i njena vrednost se smešta u promenljivu *validation*. Ako je verifikacija čvora uspešna tj. *if validation=true*, poziva se funkcija ***brisiCvor(dn).***

```
function brisiCvor(dn){

    var toDelete = dn-1;

        nodes.splice(toDelete,1);

        //poziva funkciju koja brise sve elemente svg-a
        DeleteAll();

        console.log("lines je",lines);
        // brisanje linka povezanog sa nodom
        var indeksi = [];
        for(i=0; i<lines.length; i++){
            var linki = lines[i];
            if((linki.n1 == dn) ||(linki.n2 == dn)){
                indeksi.push(i);
                console.log("proverava da li ima",dn," :",linki.n1,linki.n2);
            }
        }
        console.log("indexi",indeksi);

        var temp = [];
        for(i=0;i< lines.length;i++){
            console.log("index of",indeksi.indexOf(i));
            if(indeksi.indexOf(i)==-1){
                var linki = lines[i];
                temp.push(linki);
            }
        }
```

```
        console.log("temp",temp);

        lines = temp;
        for(i=0;i<lines.length;i++){
            var linki = lines[i];
            if(linki.n1 > dn){
                linki.n1 = (linki.n1 -1);
            }
            if(linki.n2 > dn){
                linki.n2 = (linki.n2 -1);
            }
        }
        // brisanje toka povezanog sa nodom
        var indeksi = [];
        for(i=0; i<flows.length; i++){
            var flowi = flows[i];
            if((flowi.n1 == dn) ||(flowi.n2 == dn)){
                indeksi.push(i);
                console.log("proverava da li ima",dn," :",flowi.n1,flowi.n2);
            }
        }
        console.log("indexi",indeksi);

        var temp = [];
        for(i=0;i< flows.length;i++){
            console.log("index of",indeksi.indexOf(i));
            if(indeksi.indexOf(i)==-1){
                var flowi = flows[i];
                temp.push(flowi);
            }
        }

        console.log("temp",temp);

        flows = temp;
        for(i=0;i<flows.length;i++){
            var flowi = flows[i];
            if(flowi.n1 > dn){
                flowi.n1 = (flowi.n1 -1);
            }
            if(flowi.n2 > dn){
                flowi.n2 = (flowi.n2 -1);
            }
        }


        console.log("array linije ispravljen",lines);
        console.log("array toka ispravljen",flows);

        DrawAll();

}
```

Korisnik u polje za unos podataka unosi redni broj čvora u mrežnoj topologiji, a indeksiranje nizova u *JavaScript* jeziku počinje od nule, pa da bi se dobio indeks elementa niza *nodes* koji se treba izbrisati, uneti redni broj čvora za brisanje se mora umanjiti za jedan. Ova vrednost se smešta u lokalnu promenljivu, *toDelete*. Kao i kod brisanja linka primenjujemo metodu ***splice()***. Nakon toga se poziva funkcija koja briše sve elemente niza, ***DeleteAll().*** Pošto su čvorovi povezani linkovima, samo brisanjem čvorova dobili bismo lebdeće linije pa je potrebno i linkove obrisati. Nakon toga, formiramo niz *indeksi* koji sadrži članove niza *lines* koji će se izbaciti jer povezuju čvorove koji se brišu. U ovaj niz će biti smešteni oni indeksi niza *Lines* koji odgovaraju elementima čiji je početni ili krajnji čvor redni broj čvora koji se treba obrisati, a to je *dn*. Elementi niza indeksi se određuju prolazom kroz elemente niza *Lines*, pomoću for petlje. U svakom prolazu ispituje se da li dati link ima kao početni ili krajnji čvor *dn* i ako ima, indeks tog linka u nizu *Lines* se dodaje nizu *indeksi*. Dodavanje se obavlja pomoću metode ***push()***. Ovim smo dobili niz indeksa elemenata koje treba izbrisati iz niza *lines*. Suština ovog postupka je da dobijemo niz *lines* bez elemenata koji predstavljaju elemente niza *indeksi*, pa taj ažuriran niz *lines* se dobija građenjem novog niza. To postižemo for petljom. Formira se niz *temp*. U svakom prolazu ispituje se da li je vrednost indeksa *i* član niza *indeksi* pomoću metode ***indexOf()***. Ova metoda pretražuje zadatu vrednost u nizu i vraća njenu poziciju. Ukoliko nema rezultata pretrage, u ovom slučaju ukoliko se *i* ne nalazi u nizu *indeksi*, vraća -1. U tom slučaju će se element niza *lines* sa indeksom *i* dodati na kraj niza *temp* i po izlasku iz for petlje, *temp* će predstavljati niz *lines* bez linkova koji su povezani sa čvorom *dn*. Potrebno je još niz *temp* pridružiti globalnom nizu *lines*. I na kraju je potrebno još ažurirati redne brojeve čvorova tako da bisanjem određenog čvora se ne izgubi pravilan rast rednih brojeva. For petljom prolazimo kroz svaki član niza *lines* i smeštamo ga u link *i*. Proveramo da li su redni brojevi početnog i krajnjeg čvora veći od čvora *dn* koji smo obrisali. Ako jesu, njihova vrednost se umanjuje za 1.

Identičan postupak se primenjuje i za tokove koji se odnose na dati čvor. Na kraju se poziva funkcija ***DrawAll().***

Funkcija koja briše tokove naziva se ***deleteFlow().*** HTML kod korišćen za ovaj deo je:

```
<h2>Brisanje toka</h2>
            <p>Početni čvor:<input type="text" id="df1"></p>
            <p>Krajnji čvor:<input type="text" id="df2"></p>
            <button id="deleteFlow" type="button" onclick="deleteFlow()">obriši
tok</button>

function deleteFlow(){

    var d1 = document.getElementById("df1").value;
    var d2 = document.getElementById("df2").value;
    var d1 = parseInt(d1);
    var d2 = parseInt(d2);
    var el = "tok";
    var niz = flows;
    brisi(d1,d2,el,niz);
}
```

U funkciji ***deleteFlow()*** uvodimo promenljive *d1* i *d2* koje se odnose na početni i krajnji čvor toka i one su *integer* tipa. Takođe, formiramo *el* i niz *niz* koji se odnose na tok i niz flows. Poziva se funkcija ***brisi().***

```
function brisi(d1,d2,el,niz){
    var deleteElementInArray = false;
    var toDelete;
```

```
    for(j=0;j<niz.length;j++){
        var nizovi = niz[j];
        // za svaki niz u lines proverava da li postoji link sa pocetni i
kranjim cvorom koji je unet da se obrise

        if(((nizovi.n1 == d1) && (nizovi.n2 == d2))||((nizovi.n1 == d2)
&&(nizovi.n2 == d1))){
            //ako postoji onda je varijabla true i uzima broj tog linka iz niza
            deleteElementInArray = true;
            toDelete = j;
            console.log("to delete: ",toDelete);
        }
    }

    console.log("d1 i d2",d1,d2);
    console.log("deletelink :",deleteElementInArray);

    if(((d1<0)||(d1>nodes.length))||((d2<0)||(d2>nodes.length))||(d1 ==
d2)||(deleteElementInArray == false)){
        // ako unete vrednosti nisu validne izbacuje obavestenje
        alert("Proverite da li "+el+" koji zelite da izbrisete postoji");
        console.log("nije obrisan",el,niz.length);
    }else{
        // ako jesu brise se linija iz niza
        console.log("pre brisanja >",el,niz.length);

        niz.splice(toDelete,1);

        console.log("posle brisanja >",el,niz.length);

        BrisiElementHtml(toDelete,el);
    }
}
```

Funkcija *brisi()* ima ulazne parametre koje smo kreirali u funkciji *deleteFlow()*, početni i krajnji čvor, tok i niz *flows*. Promenljiva *deleteElementInArray* se postavlja na vrednost *false*. Pomoću ove promenljive i uz pomoć for petlje proverava se da li stvarno postoji tok koji je zadat preko korisničkog interfejsa sa početnim i krajnjim čvorom kao tok koji treba da se obriše. Ako postoji, onda se smešta u promenljivu *toDelete*, a varijabla uzima vrednost *true* i uzima broj tog toka iz niza. Nakon toga se validiraju uneti podaci. Ako unete vrednosti nisu validne, tj. korisnik za čvorove uneo brojeve koji su manji od 0 ili koji su veći od dužine niza *nodes* ili pak tok između njih ne postoji izbacuje obaveštenje “Proverite da li tok koji zelite da izbrisete postoji”, a ako jesu briše se linija iz niza. Pošto brisanje toka podrazumeva uklanjanje elementa niza sa određenim indeksom, koristimo metodu *splice(). niz.splice(toDelete,1)* - ovo znači da se briše jedan element počevši od elementa *toDelete*. Na ovaj način je dobijen niz *niz*, sa istim redosledom elemenata, samo bez toka koji je korisnik zadao da treba da bude izbrisan. Poziva se funkcija ***BrisiElementHtml(toDelete,el).***

Ostalo je još da objasnimo funkciju ***DeleteAll()*** koja briše sve elemente svg-a kao i ***DrawAll()*** koja crta sve elemente svg-a. u funkciji *DeleteAll* formiramo promenljivu *c* koja se odnosi na sve svg elemente. U promenljivu *j* smeštamo broj svg elemenata koji smanjujemo za 1. I brišemo ga.

```
function DeleteAll(){
         //brise sve elemente svg-a
        var c = svgWindow.childNodes;
```

```
        var j= c.length -1;

        //petlja za live array
        for(j;j >=0;--j){
            var s = svgWindow.childNodes[j];
            // ako je node element svg liste
           if(s.nodeType === 1){
            svgWindow.removeChild(s);
            }
        }
}

function DrawAll(){
        // stavlja da nodovi nisu nacrtani
        for(i=0;i<nodes.length;i++){
            var nodesi = nodes[i];
            nodesi.isNodeDrawn = false;
        }

        // stavlja da linkovi nisu nacrtani
        for(i=0;i<lines.length;i++){
            var linesi = lines[i];
            linesi.isLineDrawn = false;
        }
        // stavlja da tokovi nisu nacrtani
        for(i=0;i<flows.length;i++){
            var flowsi = flows[i];
            flowsi.isFlowDrawn = false;
        }

        // crta sve elemente svg-a
        drawNodes();
        drawLines();
        drawFlows();
}
```

U funkciji ***DrawAll()*** prolazimo kroz sve elemente niza *nodes* i smeštamo ih u promenljivu ***nodesi*** , postavljamo vrednost *false*, tj. da čvorovi nisu nacrtani. Potom, sve isto uradimo za linkove i tokove. I na kraju pozovemo funkcije koje iscrtavaju sve ove elemente, ***drawNodes(), drawLines()*** i ***drawFlows().***

4.4. AŽURIRANJE ČVORA, LINKA I TOKA

Korisnik preko korisničkog interfejsa u 3. koloni može da ažurira čvorove. HTML kod koji je korišćen za izradu tog dela je:

```
<h2>Azuriraj parametre čvora:</h2>

            <p>Čvor : <input type="text" id="un"></p>
            <p>Koordinata X:<input type="text" id="ux"></p>
            <p>Koordinata Y:<input type="text" id="uy"></p>
            <p>Kašnjenje(ms):<input type="text" id="uz"></p>
            <button type="button" onclick="UpdateNode()" >azuriraj čvor</button>
```

Da bismo izvršili ažuriranje čvora, koristili smo funkciju ***UpdateNode().***

```
function UpdateNode(){
    var un = document.getElementById("un").value;
    var nx = document.getElementById("ux").value;
    var ny = document.getElementById("uy").value;
    var nz = document.getElementById("uz").value;
    var isNodeDrawn = false;

    var un = parseInt(un);
    var nx = parseInt(nx);
    var ny = parseInt(ny);
    var nz = parseFloat(nz);

    var validation = ValidateUpdateNodeParameters(un ,nx, ny, nz);
    console.log("Validate Update NODE je :",validation);
            if (validation === true){
                // ako je validacija uspesna, definise se novi cvor i upisuje
umesto
                // njega cvor sa novim koordinatama i kasnjenjem
                var node = new defineNode(nx ,ny, nz, isNodeDrawn);
                var num = un - 1;
                nodes[num] = node;
                console.log("update node koji se upisuej je: ",node);

                // za linkove koji imaju taj cvor upisuje koordinate azuriranog
cvora
                for(i=0;i < lines.length;i++){
                    var linei = lines[i];
                    if(linei.n1 == un){
                        linei.n1 = un;
                        linei.startNodeX = nx;
                        linei.startNodeY = ny;
                    }
                    if(linei.n2 == un){
                        linei.n2 = un;
                        linei.endNodeX = nx;
                        linei.endNodeY = ny;
                    }
                }

                // za tokove koji imaju taj cvor upisuje koordinate azuriranog
cvora
                for(i=0;i < flows.length;i++){
                    var flowi = flows[i];
                    if(flowi.n1 == un){
                        flowi.n1 = un;
                        flowi.startFlowX = nx;
                        flowi.startFlowY = ny;
                    }
                    if(flowi.n2 == un){
                        flowi.n2 = un;
                        flowi.endFlowX = nx;
                        flowi.endFlowY = ny;
                    }
                }

                //brise sve elemente svg-a
                DeleteAll();

                //ponovo crta sve elemente
```

```
            DrawAll();

        } else{
            console.log("Validation NODE ERROR");
        }
}
```

Funkcija uzima vrednosti parametara pomoću *document.getElementById* i kreira promenljive *un*, *nx*, *ny* i *nz*. Zatim ih konvertuje u *integer* i *float* tip i postavlja boolean na *false* tj. čvor nije nacrtan. Pozivamo funkciju *ValidateUpdateNodeParametars()* koja ima ulazne parametre *un*, *nx*, *ny*, *nz* i vrednost joj stavljamo u promenljivu *validation* koja je tipa *boolean*. Ako je validacija uspešna, definiše se novi čvor i upisuje umesto njega sa novim koordinatama i kašnjenjem. Kreiramo promenljivu *num* u kojoj skladištimo vrednost promenljive *un* koja 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 *un* postaje jednaka vrednosti indeksa tog čvora, koji se ažurira u nizu. U drugoj naredbi se elementu sa tim indeksom u nizu objekata *nodes* dodeljuje objekat *node*, koji sadrži ažurirane osobine objekta čvora.

Zatim, sledi deo koji se odnosi na linkove koji povezuju čvor koji se ažurira. For petljom prolazimo kroz niz *lines* i smeštamo svaki član tog niza u promeljivu *linei*. Ako je njegova vrednost *n1* identična kao vrednost promenjive *un* onda se varijablama tog elementa niza *linei.n1* dodeljuju vrednosti koje smo hteli da ažuriramo u html-u.

Identičan postupak primenjujemo i za tokove koji se odnose na ažurirani čvor. Na kraju imamo dve funkcije ***DeleteAll()*** i ***DrawAll()*** koje brišu i crtaju sve, respektivno.

Pored ažuriranja čvorova, korisnik može da ažurira i linkove. HTML kod korišćen u ovom slučaju je:

```
<h2>Ažuriraj parametre linka:</h2>

            <p>Pocetni čvor linka :<input type="text" id="uln1"></p>
            <p>Krajnji čvor linka :<input type="text" id="uln2"></p>
            <p>Cena :<input type="text" id="c"></p>
            <p>Kašnjenje(ms):<input type="text" id="ulk"></p>
            <p>Kapacitet(kb/s):<input type="text" id="ulkl"></p>
            <button type="button" onclick="UpdateLink()" >ažuriraj link</button>
```

Funkcija za ažuriranje linkova je ***UpdateLink().***

```
function UpdateLink(){
    var uln1 = document.getElementById("uln1").value;
    var uln2 = document.getElementById("uln2").value;
    var ulk = document.getElementById("ulk").value;
    var ulkl = document.getElementById("ulkl").value;
    var c = document.getElementById("c").value;
    var isNodeDrawn = false;

    var uln1 = parseInt(uln1);
    var uln2 = parseInt(uln2);
    var ulk = parseFloat(ulk);
    var ulkl = parseFloat(ulkl);
    var c = parseInt(c);


    var validation = ValidateUpdateLine(uln1, uln2,c, ulk, ulkl);

    console.log("Validate Update Link je :",validation);
```

```
            if (validation === true){
                // ako je verifikacija uspesna trazi se element niza koji sadrzi
                // pocetni i krajnji cvor i azurira kasnjenje linka i kapacitet
linka
                for(i=0;i < lines.length;i++){
                  var linesi = lines[i];

                  if(((linesi.n1 == uln1)||(linesi.n1 == uln2))&&((linesi.n2 ==
uln1)||(linesi.n2 == uln2))){
                        linesi.kasnjenjeLinka = ulk;
                        linesi.kapacitetLinka = ulkl;
                        linesi.cenaLinka = c;


                        console.log("azuriranje linka[",i,"] je uspelo");

                        console.log("azuriran link :",linesi);


                     }
                }

            } else{
                console.log("Validation NODE ERROR");
            }
}
```

Funkcija uzima vrednosti parametara pomoću *document.getElementById* i kreira promenljive *uln1*, *uln2*, *c*, *ulk* i *ulk1*. Zatim ih konvertuje i postavlja boolean na *false* tj. link nije nacrtan. Pozivamo funkciju *ValidateUpdateLine()* koja ima ulazne parametre *uln1*, *uln2*, *c*, *ulk*, *ulk1* i vrednost joj stavljamo u u promenljivu *validation*. Ako je verifikacija uspešna traži se element niza koji sadrži početni i krajnji čvor linka koji je jednak vrednostima koje ažuriramo, a ažuriramo i kašnjenje linka i kapacitet linka.

I za kraj je potrebno ažurirati tok. HTML kod za ažuriranje toka je:

```
<h2>Ažuriraj parametre toka:</h2>

            <p>Pocetni čvor toka :<input type="text" id="ufn1"></p>
            <p>Krajnji čvor toka :<input type="text" id="ufn2"></p>
            <p>Kapacitet(kb/s):<input type="text" id="ufk"></p>
            <button type="button" onclick="UpdateFlow()" >ažuriraj tok</button>
```

Funkcija za ažuriranje toka je ***UpdateFlow().***

```
function UpdateFlow(){
    var ufn1 = document.getElementById("ufn1").value;
    var ufn2 = document.getElementById("ufn2").value;
    var ufk = document.getElementById("ufk").value;
    var isNodeDrawn = false;

    var ufn1 = parseInt(ufn1);
    var ufn2 = parseInt(ufn2);
    var ufk = parseFloat(ufk);

    var validation = ValidateUpdateFlow(ufn1, ufn2, ufk);
```

```
    console.log("Validate Update Flow je :",validation);

            if (validation === true){
                // ako je verifikacija uspesna trazi se element niza koji sadrzi
                // pocetni i krajnji cvor i azurira kasnjenje toka
                for(i=0;i < flows.length;i++){
                  var flowi = flows[i];

                  if(((flowi.n1 == ufn1)||(flowi.n1 == ufn2))&&((flowi.n2 ==
ufn1)||(flowi.n2 == ufn2))){

                        flowi.kasnjenjeToka = ufk;

                        console.log("azuriranje toka[",i,"] je uspelo");

                        console.log("azuriran tok ",flowi);
                   }
                }

            } else{

                // ako verifikacija toka nije uspesna nista se ne desava,
predhodno u validation.js izbacuje obavestenje
                // koja je greska u pitanju
                console.log("Validation FLOW ERROR");
            }
}
```

Tok funkcije je isti kao u prethodna dva slučaja, za čvor i link. Ako je verifikacija uspešna traži se element niza koji sadrži početni i krajnji čvor i ažurira kašnjenje toka. Ako verifikacija toka nije uspešna ništa se ne dešava.

4.5. VALIDACIJA 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 *nodes*, *lines i flows*, potrebno je proveriti njihovu ispravnost. Funkcije za proveravanje tačnosti unetih parametara kao ulazne parametre imaju parametre čvora, linka ili toka, a kao izlazni parametar vrednost *boolean* promenljive *valid* koji ima vrednost “true“, ako su parametri ispravni, odnosno “false“, u suprotnom slucaju.

Funkcija za proveru tačnosti parametara čvora je ***ValidateNodeParameters().***

```
function ValidateNodeParameters(nx,ny,nz){

        var x = String(nx);
        var y = String(ny);
        var z = String(nz);
        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
        // okir svg-a
        var svgWindow = document.getElementById("mySvg");
        var svgWindowWidth = parseInt(svgWindow.style.width);
        var svgWindowHeight = parseInt(svgWindow.style.height);

        if((x.match(patt1))||(y.match(patt1))||(z.match(patt2))||(x===null ||
x==="")||(y===null || y==="")||(z===null || z===="")){
```

```
            // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

            valid = false;
            alert("Parametri cvora ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
        }else{
            if ((nx<20)||(nx>svgWindowWidth-20)||(ny<20)||(ny>svgWindowHeight-
20)){
            // ako tacka izlazi iz okvira svg -20 se dodaje da bi ceo krug mogao
da stane a ne pola, ako je na ivici
            valid = false;
            alert("Uneli ste pogresne koordinate cvora, van okvira prozora za
crtanje !");
        }else{
            if(nodes.length>0){
                // ako je unet bar jedan cvor
                var N = nodes.length;
                for(i=0;i<N;i++){
                    var node = nodes[i];
                    if(node.nx == x){
                        if(node.ny == y){
                            // proverava dali vec postoji zadati cvor

                            valid = false;
                            alert("Dati cvor vec postoji !");
                        }
                    }
                }
            }}}

        //vraca valid true ako je funkcija prosla validaciju, u suprotnom false
        return valid;
    }
```

Postavlja se promenljiva *valid* na vrednost *true*. Prvo što treba proveriti jeste da li uneti podaci sadrže slova ili specijalne karaktere. Zbog toga uvodimo *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. Uvodimo i parametre koji se odnose na okvir SVG-a. Zatim, uslovom if se proverava ako parametri sadrže specijalna slova, razmak i specijalne karaktere ili su prazni. Kako su koordinate tipa *string*, parametri *x, y i z* mogu da sadrže tačku. Ukoliko postoji nešto iz uslova, *valid* se postavlja na vrednost *false* i korisnik se obaveštava o grešci porukom pomoću alert "Parametri čvora ne smeju sadržati slova, razmak, ili specijalne karaktere! ".Ukoliko je korisnik uneo numeričke podatke za koordinate čvora, proverava se da li tačka izlazi iz okvira svg. Dodaje se -20 da bi ceo krug mogao da stane, a ne pola, ako se krug nalazi kojim slučajem na ivici. Tada se *valid* postavlja na vrednost "*false*" i korisnik se obaveštava o grešci porukom "Uneli ste pogresne koordinate čvora, van okvira prozora za crtanje !". Zatim se vrši provera da li zadati čvor već postoji u nizi *nodes*. U slučaju da postoji, korisnik je obavešten porukom "Dati čvor već postoji !".

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(un,nx,ny,nz){

        var un = String(un);
        var n = parseInt(un);
        var x = String(nx);
        var y = String(ny);
        var z = String(nz);
        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
        // okir svg-a
        var svgWindow = document.getElementById("mySvg");
        var svgWindowWidth = parseInt(svgWindow.style.width);
        var svgWindowHeight = parseInt(svgWindow.style.height);

        if((un.match(patt1)) ||
(x.match(patt1))||(y.match(patt1))||(z.match(patt2))||(un === null ||
un===""||(x===null || x===""||(y===null || y===""||(z===null || z===""){
            // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

            valid = false;
            alert("Parametri cvora ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
        }else{

            console.log("n je ",n,"length",nodes.length);
            if((n < 0)||(n>nodes.length)){
                alert("Cvor koji se azurira ne postoji");
                valid = false;
            }else{
                if ((nx<20)||(nx>svgWindowWidth-
20)||(ny<20)||(ny>svgWindowHeight-20)){
                // ako tacka izlazi iz okvira svg -20 se dodaje da bi ceo krug
mogao da stane a ne pola, ako je na ivici
                valid = false;
                alert("Uneli ste pogresne koordinate cvora !");
                }else{
                    if(nodes.length>0){
                    // ako je unet bar jedan cvor
                    var N = nodes.length;
                    for(i=0;i<N;i++){
                        var node = nodes[i];
                        if(node.nx == x){
                            if(node.ny == y){
                            // proverava dali vec postoji zadati cvor

                            valid = false;
                            alert("Dati cvor vec postoji !");
                        }
                    }
                    }
                    }   }
            }
            }

        //vraca valid true ako je funkcija prosla validaciju, u suprotnom false
        return valid;
    }
```

Razlika u odnosu na prvu funkciju je što je sada potrebno odrediti redni broj čvora. Na početku funkcije se deklarišu promenljive *valid*, koja je izlazni parametar funkcije, i *patt1* i *patt2*. Uzorak *patt2* obuhvata i pojavljivanje tačke, koje ne sme biti u *integer* tipu podataka, u ovom slučaju promenljiva *n*. Zatim uslovom if se proverava ako parametri sadrže specijalna slova, razmak i specijalne karaktere ili su prazni. 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 imaju smisla. Tako su pokriveni slucajevi da korisnik za redni broj cvora unese broj manji od jedinice ili broj veći od ukupnog broja čvorova, tada korisnik dobija poruku "Čvor koji se ažurira ne postoji!". Zatim, za slučaj kada nove koordinate određuju tačku van prostora *svg* elementa i slucaj kada čvor sa zadatim novim koordinatama već postoji u topologiji, odnosno u nizu *nodes*. Za sva tri slučaja *valid* se postavlja na "*false*", a korisniku se prikazuje adekvatna poruka o grešci.

Proveru ispravnosti unetih parametara linka vrši funkcija ***ValidateLinkParameters().***

```
function ValidateLineParameters(n1, n2,cenaLinka,
kasnjenjeLinka,kapacitetLinka){

        var x = String(n1);
        var y = String(n2);
        var z = String(kasnjenjeLinka);
        var w = String(kapacitetLinka);
        var c = String(cenaLinka);
        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;

        var N = nodes.length;
        var L = lines.length;


        if (N < 2){
            // ako nije unet nijedan cvor, ili samo jedan cvor
            valid = false;
            alert("Mora se uneti minimum dva cvora");
            console.log("N je",N);
        }else{


if((x.match(patt1))||(y.match(patt1))||(z.match(patt2))||(w.match(patt2)||(c.mat
ch(patt1)))
                    ||(x===null || x==="")||(y===null || y==="")
                    ||(z===null || z==="")||(w===null || w==="")
                    ||(c===null || c==="")){

                // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

                valid = false;
                alert("Parametri linka ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
            }else{
```

```
            if(x == y){
            // unet je isti cvor za pocetni i kranji cvor

                valid = false;
                alert("Cvorovi koje povezuje link moraju biti razliciti");
            }else{

                if ((x < 1)||(y < 1)||(x > N)||(y > N)){
                  // ako cvor sa unetim rednim broj ne postoji
                    valid = false;
                    alert('Proverite da li uneti cvor za link postoji');
                }else{

                for(i=0;i<L;i++){
                    var linesi = lines[i];
                    if((linesi.n1 == x)||(linesi.n1 == y)){
                        if((linesi.n2 == y)||(linesi.n2 == x)){

                            // ako novi link vec postoji u nizy lines
                            valid =false;
                            alert("Link vec postoji !");
                        }
                    }
                }}
    }}}

    // vraca valid
    return valid;

        }
```

Funkcija *ValidateLineParameters()* ima za ulazne parametre početni čvor linka *n1*, krajnji čvor linka *n2* , cenu, kapacitet i kašnjenje linka. Izlazni parametar je *valid*, koji ima vrednost "*true*", ako su ispunjeni zadati uslovi validnosti. Definiše se promenljiva *N* koja predstavlja dužinu niza *nodes* kao i *L* za dužinu niza *lines*. U slučaju da nije unet nijedan čvor, ili samo jedan čvor, funkcija vraća vrednost *false* i korisnik dobija poruku "Mora se uneti minimum dva čvora". U drugom slučaju, ako parametri sadrže specijalna slova, razmak i specijalne karaktere ili su prazni, funkcija ponovo vraća vrednost *false* i korisnik dobija poruku "Parametri linka ne smeju sadrzati slova, razmak, ili specijalne karaktere !". Ako su početni i krajnji čvorovi zapravo isti, vraća vrednost *false* i korisnik dobija poruku "Cvorovi koje povezuje link moraju biti razliciti". Ako čvor sa unetim rednim brojem ne postoji, tj. ako ne pripada nizu *lines*, dobija se poruka 'Proverite da li uneti čvor za link postoji'. I na kraju, ako novi link već postoji u nizu *lines*, poruka za korisnika je "Link već postoji !".

Funkcija koja proverava ispravnost unetih podataka prilikom ažuriranja linka je ***ValidateUpdateLinkParameters()***

```
function ValidateUpdateLine(n1, n2,c, ulk, ulkl){

    var x = String(n1);
    var y = String(n2);
    var z = String(ulk);
    var w = String(ulkl);
    var c = String(c);
    var valid = true;
    var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
```

```
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;

        var N = nodes.length;
        var L = lines.length;



if((x.match(patt1))||(y.match(patt1))||(z.match(patt2))||(w.match(patt2)||(c.mat
ch(patt1)))
                    ||(x===null || x==="")||(y===null || y==="")
                    ||(z===null || z==="")||(w===null || w==="")
                    ||(c===null || c==="")){

                // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

                valid = false;
                alert("Parametri linka ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
            }else{


                if(x == y){
                // unet je isti cvor za pocetni i kranji cvor

                    valid = false;
                    alert("Cvorovi koje povezuje link moraju biti razliciti !");
                }else{

                    if ((x < 1)||(y < 1)||(x > N)||(y > N)){
                      // ako cvor sa unetim rednim broj ne postoji
                        valid = false;
                        alert('Proverite da li uneti cvor za link postoji !');
                    }else{
                        // proverava da li postoje cvorovi u nizu linkova, ako
nema onda nije uspesna verifikacija
                        var linkInLinkovi = true ;
                        for(i=0;i<L;i++){
                            var linki = lines[i];
                            if(((linki.n1 == x)&&(linki.n2 ==y))||((linki.n2 ==
x)&&(linki.n1 == y))){
                                linkInLinkovi = false;

                            }

                        }

                        if(linkInLinkovi === true){
                            valid = false;
                            alert("Proverite da li link koji ste azurirali
postoji !");
                        }
                    }
        }}
        // vraca valid
        return valid;

            }
```

Parametri *x*, *y*, *z*, *w*, *c* se odnose na početni i krajnji čvor linka, na novu cenu, kašnjenje i kapacitet linka. Ako parametri sadrže specijalna slova, razmak i specijalne karaktere ili su prazni, ako unet je isti čvor za početni i kranji čvor, ako čvor sa unetim rednim broj ne postoji, funkcija vraća vrednost *false* i korisnik dobija određene poruke obaveštenja. Zatim, funkcija još proverava da li postoji link u nizu *lines*.

Poslednja stvar u ovom poglavlju koju je potrebno objasniti je funkcija za validaciju toka tj. ***ValidateFlowParameters().***

```
function ValidateFlowParameters(n1,n2,kapacitetToka){

        var x = String(n1);
        var y = String(n2);
        var z = String(kapacitetToka);

        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;

        var N = nodes.length;
        var F = flows.length;

        if (N < 2){
            // ako nije unet nijedan cvor, ili samo jedan cvor
            valid = false;
            alert("Mora se uneti minimum dva cvora");
        }else{

            if((x.match(patt1))||(y.match(patt1))||(z.match(patt2))
                ||(x===null || x==="")||(y===null || y==="")
                ||(z===null || z==="")){

            // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

            valid = false;
            alert("Parametri toka ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
        }else{

            if(x == y){
            // unet je isti cvor za pocetni i kranji cvor

            valid = false;
            alert("Cvorovi toka moraju biti razliciti");
            }else{

                if ((x < 1)||(y < 1)||(x > N)||(y > N)){
                // ako cvor sa unetim rednim broj ne postoji
                    valid = false;
                    alert('Proverite da li uneti cvor za tok postoji');
                }else{

                for(i=0;i<F;i++){
                     var flowsi = flows[i];
                     if((flowsi.n1 == x)||(flowsi.n1 == y)){
```

```
                        if((flowsi.n2 == y)||(flowsi.n2 == x)){
                        // ako novi link vec postoji u nizy lines

                            valid =false;
                            alert("Tok vec postoji !");
                        }
                    }
                }
        }}}}
        // vraca valid
        return valid;
    }
```

Ova funkcija ima za ulazne parametre početni i krajnji čvor, *n1* i *n2*, respektivno i parameter za kapacitet toka. Promenljiva koja se deklariše u telu funkcije je *valid*. Ona je izlazni parameter 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. Postoje uslovi koji proveravaju 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 "Parametri toka ne smeju sadrzati slova, razmak, ili 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 *nodes* i na kraju sledi provera da li novi tok koji je definisan krajnjim čvorovima *x* i *y* već postoji u nizu *flows*. 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.

Kada korisnik želi da ažurira parametara toka, on zadaje početni i krajnji čvor toka koji se ažurira, pomoću kojih se identifikuje zadati tok u nizu *flows*, i unosi novu vrednost kapaciteta. Funkcija koja proverava ispravnost unetih podataka prilikom ažuriranja toka je ***ValidateUpdateFlowParameters()***.

```
function ValidateUpdateFlow(n1, n2, ulk){

        var x = String(n1);
        var y = String(n2);
        var z = String(ulk);

        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;

        var N = nodes.length;
        var L = flows.length;


            if((x.match(patt1))||(y.match(patt1))||(z.match(patt2))
                    ||(x===null || x==="")||(y===null || y==="")
                    ||(z===null || z==="")){

                // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni
```

```
                valid = false;
                alert("Parametri toka ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
            }else{

                if(x == y){
                // ako unet je isti cvor za pocetni i kranji cvor toka

                    valid = false;
                    alert("Cvorovi koje povezuje tok moraju biti razliciti !");
                }else{

                    if ((x < 1)||(y < 1)||(x > N)||(y > N)){
                      // ako cvor sa unetim rednim broj ne postoji
                        valid = false;
                        alert('Proverite da li uneti cvor za tok postoji !');
                    }else{
                        var flowInFlows = true ;

                        // proverava da li za unete cvorove postoji tok koji ce
da se azurira
                        for(i=0;i<L;i++){
                            var flowi = flows[i];
                            if(((flowi.n1 == x)&&(flowi.n2 ==y))||((flowi.n2 ==
x)&&(flowi.n1 == y))){

                                flowInFlows = false;

                            }

                        }

                        if(flowInFlows === true){
                            valid = false;
                            alert("Proverite da li tok koji ste azurirali
postoji !");
                        }
                    }
        }}

        // vraca valid
        return valid;

            }
```

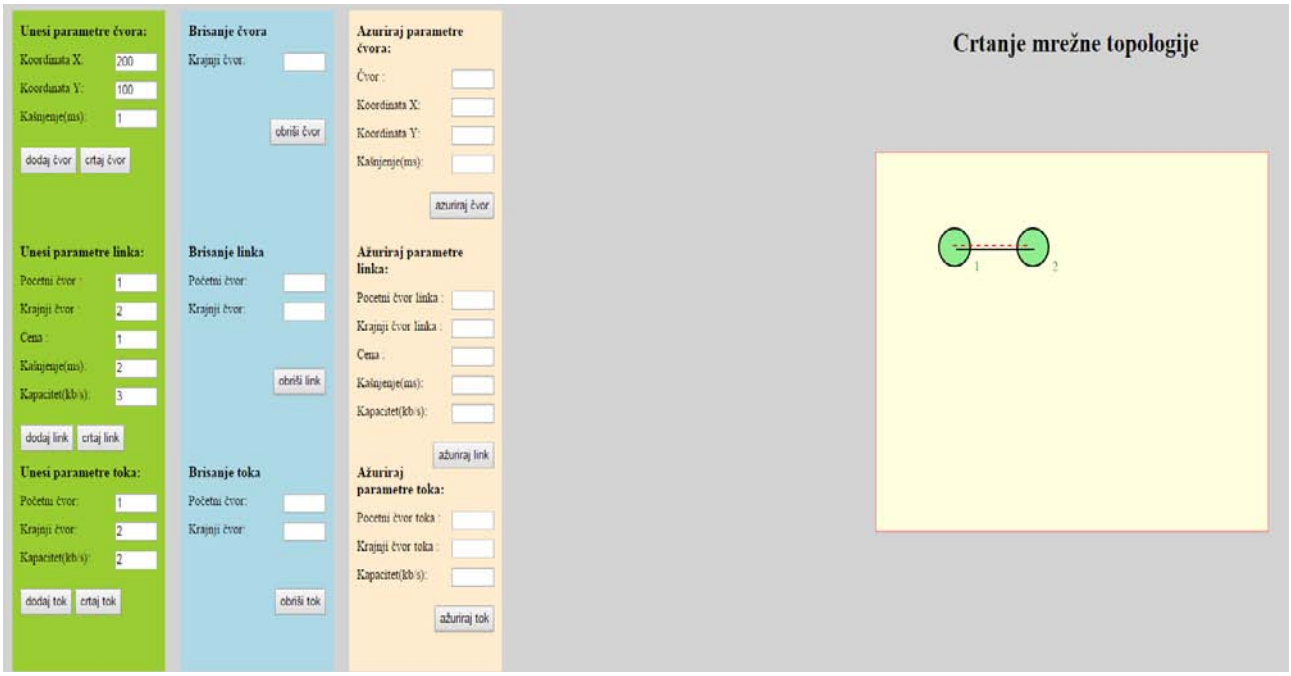
Njeni ulazni parametri su početni i krajnji čvor toka i kapacitet toka. Promenljiva koja se deklariše u telu funkcije je *valid.* Ona je izlazni parameter funkcije i ima vrednost "*true*", ako je korisnik uneo ispravne podatke, odnosno "*false*" u suprotnom slučaju. Zatim se deklarišu uzorci *patt1* 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 *patt1* ne proverava pojavljivanje i tačke, koju korisnik može uneti u polje za kapacitet toka. Postoje uslovi koji proveravaju 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 "Parametri toka ne smeju sadrzati slova, razmak, ili 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 *nodes* i na kraju sledi provera da li novi tok koji je definisan krajnjim čvorovima *x* i *y* već postoji u nizu flows. 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.

Potrebno je još dodati funkciju koja proverava ispravnost unetih patametar za čvor koji korisnik želi da obriše, odnosno ***ValidateDeleteNodeParameters().***

```
function ValidateDeleteNodeParameters(dn){

        var x = String(dn);
        var valid = true;
        var patt1 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\]:";'<>?\//\s]/i;
        var patt2 = /[A-Z-!$%^&*()_+|~=`\\#{}\[\].:";'<>?\//\s]/i;
        // okir svg-a
        var svgWindow = document.getElementById("mySvg");
        var svgWindowWidth = parseInt(svgWindow.style.width);
        var svgWindowHeight = parseInt(svgWindow.style.height);

        if((x.match(patt1))||(x===null || x===""){
            // ako parametri sadrze specijalna slova, razmak i specijalne
karaktere ili su prazni

            valid = false;
            alert("Parametri cvora ne smeju sadrzati slova, razmak, ili
specijalne karaktere !");
        }else{
            if((x < 0)||(x > nodes.length)){
            // ako ne postoji node
            alert("Provertite da li node koji hocete da izbrisete postoji.");
            valid = false;

            }
        }
        return valid;
    }
```

Ova funkcija ima kao ulazni parameter krajnji čvor, *dn*. Kao i dosad, tok fukcije je isti, proverava se metodom *match()* da li parametri sadrže specijalna slova, razmak i specijalne karaktere ili su prazni i da li postoji čvor. Ukoliko je neki od uslova ispunjen funkcija vraća vrednost *false*, u suprotnom *true*.



Sl.4.5.1. - Konačan izgled veb stranice

5. PRIMERI PRIMENE BIBLIOTEKE FUNKCIJA

Poslednje poglavlje u ovom radu je rezervisano za primere prethodno navedenih funkcija, tj. primer za ručno unošenje i crtanje elemenata mrežne topologije. Radi lakšeg manipulisanja koristićemo *jquery*.

Na našoj HTML stranici, na samom startu smo definisali svg prozor, širine i dužine 500px i 400px, respektivno, koristeći <style> za njegovo stilizovanje na sledeći način:

```
#mySvg{
    background: lightyellow;
    border:1px solid lightcoral;
    position:relative;
    width:500px;
    height:400px;
    top:40px;
    margin: 40px 50px 0px 40px ;

}
```

5.1. RUČNO UNOŠENJE I CRTANJE ELEMENATA

Pod pojmom "ručno unošenje i crtanje elemenata" podrazumeva se da kada kliknemo bilo gde na našem *svg* prozoru, u polju označenom kao koordinata X i koordinata Y, pojavljuju se upravo te koordinate, pošto su parametri naših čvorova zapravo koordinate centara kružnica. S tim u vezi potrebe su nam koordinate miša, pa zato kreiramo odgovarajuću funkciju koja vraća te vrednosti**.** Koordinata miša u odnosu na prozor dobija se pomoću *clientX* i *clientY* atributa objekta događaja koji je vezan za miš. *ClientX* je horizontalna, a *clientY* je vertikalna koordinata miša u odnosu na prozor. Da bi se dobile koordinate tačke na *svg*-u, potrebno je oduzeti horizontalnu koordinatu levog gornjeg ugla *svg*-a od horizontalne koordinate pozicije miša u prozoru i vertikalnu koordinatu levog gornjeg ugla *svg*-a od vertikalne koordinate pozicije miša u prozoru. Funkcija koja sledi vraća *x* i *y* koordinatu tačke u *svg*-u:

```
var bodyClickCount = 1;
$( "#mySvg" ).mousemove(function( event ) {
    if(bodyClickCount % 2){


  var p = $("#mySvg");
  var offset = p.offset();
  var pageCoordsX = event.pageX-(offset.left.toFixed());
  var pageCoordsY = event.pageY-(offset.top.toFixed());
  var clientCoords = "( " + event.clientX + ", " + event.clientY + " )";
  $( "#x" ).val(pageCoordsX);
  $( "#y" ).val(pageCoordsY);
    }
});

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

```
   bodyClickCount++;
});
```

Prva stvar koja se definiše *BodyClickCount*. Ona je varijabla koja broji koliko klikova je urađeno u svg elementu. Zatim smo koristili *mousemove* metod. Svaki put kada korisnik pomeri miša, tj. kada se pređe mišem preko #mySVg polja, tj. svg elementa, pokreće se cela sledeća funkcija.

Potom *if(bodyClickCount % 2)* - varijabla moduo 2, izračunava ostatak pri deljenju sa dva, tako se if grana pokreće prilikom klika. Zatim sledi deo kojim dobijamo x i y koordinate.

event.pageX ili Y daju koordinate ali od početka ekrana, ne daju koordinate od početka svg - elementa zato oduzimamo sa offset koordinatama, koje daju piksele od početka ekrana(tj. brauzera) do svg- elementa.

```
var pageCoordsX = event.pageX-(offset.left.toFixed());
var pageCoordsY = event.pageY-(offset.top.toFixed());
```

Konačno, sledećim kodom dobijamo koordinate miša u polju za unos čvora.

```
$( "#x" ).val(pageCoordsX);
$( "#y" ).val(pageCoordsY);
```

Na samom kraju stoji deo kojim postižemo da kada kliknemo unutar svg prozora povećavamo varijablu za jedan.

```
$("#mySvg").on('click',function(){
   bodyClickCount++;

});
```

5.2. PRIMENA DIJKSTRA ALGORITMA NA SELEKTOVANI ČVOR

Za početak ćemo objasniti šta je zapravo Dijkstra algoritam. Najjednostavnije objašnjenje je da je on jedan od načina za pronalaženje najkraće rute između dva čvora u grafu.

Jedan od važnijih zadataka prilikom konstruisanja mrežnog sloja su algoritmi za rutiranje, tj. pronalaženje najjednostavnije i najjeftinije putanje između dva rutera. Na ovom algoritmu su bazirani OSPF (*Open Shortest Path First*) i IS-IS (*Intermediate System to Intermediate System*) protokoli rutiranja, a i u drumskom saobraćaju je dobio veliku ulogu.

Nazovimo čvor od kog krećemo inicijalni čvor. Ovaj algoritam pamti vrednost trenutno najkraćeg puta od inicijalnog čvora za svaki čvor iz grafa, tj. stabla. Na početku se ta vrednost postavlja na nulu. I još ćemo napomenuti da se pronalazi putanja sa najmanjom cenom između tog čvora i svih ostalih u mreži.

```
function Dijkstra(){
    var radniCvor1 = document.getElementById("pocetni").value;
    var cvoroviUStablu=[];
    var linkoviUStablu=[];
    var trencena=0;
    var cene=[];
    var preth=[];
    var susedi=[];

    var radniCvor = parseInt(radniCvor1);

    console.log("radni cvor: ",radniCvor);

    cvoroviUStablu.push(radniCvor);

    console.log("cvorovi u stablu niz: ",cvoroviUStablu);

    while(cvoroviUStablu.length < nodes.length){
        var k=0;

        while(k < lines.length ){
            var linki = lines[k];
            console.log("dijkstra linki je: ",linki);

            if(radniCvor == linki.n1){
                var q = linki.n2;

                if(cvoroviUStablu.indexOf(q)< 0){
                   console.log("cvorovi index of ",q);
                   susedi.push(q);
                   console.log("susedi ",q);
                   cene.push(trencena+linki.cenaLinka);
                   console.log("cena za upisivanje u cene
:",trencena,"+",linki.cenaLinka,"=",trencena+linki.cenaLinka);
                   preth.push(radniCvor);
                   console.log("preth :",radniCvor);
                }

            }
            if(radniCvor == linki.n2){
                var q = linki.n1;

                if(cvoroviUStablu.indexOf(q) <0){
                   console.log("cvorovi index of ",q);
                   susedi.push(q);
                   console.log("susedi ",q);
                   cene.push(trencena+linki.cenaLinka);
                   console.log("cena za upisivanje u cene
:",trencena,"+",linki.cenaLinka,"=",trencena+linki.cenaLinka);
                   preth.push(radniCvor);
                   console.log("preth :",radniCvor);
                }
            }
            k++;
        }

        var min = cene[0];
        b=0;
        var i=1;
```

```
        while (i<cene.length){
            console.log("dok je ",i,"manje od",cene.length);
            if(min>cene[i]){
                console.log("ako je",min," vece od",cene[i]);
                b=i;
                console.log("varijabla B je: ",b);
            }
            i++;
        }

        for(a=0; a< lines.length;a++){
            var linki = lines[a];
            if(((linki.n1 == susedi[b])&&(linki.n2 == preth[b]))||
               ((linki.n1 == preth[b])&&(linki.n2 == susedi[b]))){

                linkoviUStablu.push(linki);
                console.log("push linki u linkove u stablu: ",linki);
              }

        }

        console.log("susedi b:",susedi[b]);
        cvoroviUStablu.push(susedi[b]);
        console.log("cvorovi u stablu niz: ",cvoroviUStablu);
        trencena = min;

        radniCvor = susedi[b];

        var susedi_out=[];

        var l=0;

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

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



    drawLines();
    drawNodes();

    }
```

Ova funkcija se poziva klikom na dugme crtaj Dijkstra. HTML kod za njegovo generisanje je:

```
<div><p>Dijkstra algoritam</br> pocetni cvor: <input type="text"
id="pocetni"></p>
        <button type="button" onclick="Dijkstra()" >crtaj Dijkstra</button>
        <span class="boxMouse1"></span>
        <span class="boxMouse2"></span>
```

Korisnik u input polje unosi čvor koji predstavlja inicijalni čvor za koji se računa najkraća putanja do svih ostalih čvorova u mreži.

Polje za unos inicijalnog čvora je označeno kao početni čvor i njegova vrednost se postavlja u promenljivu *RadniCvor*. Ova promenljiva će kroz funkciju postati poslednji čvor koji je dodat u stablo, odnosno poslednji čvor do kojeg je određena najbolja putanja od inicijalnog čvora. Taj čvor će u daljem tekstu biti nazvan tekući radni čvor, čvorovi do kojih je određena najbolja putanja pre tekućeg radnog čvora neka se zovu prethodni radni čvorovi. *CvoroviUStablu* je definisan kao prazan niz u koji će se dodavati čvorovi topologije, tokom prolaska kroz petlju. *LinkoviUStablu* odnosi se na niz linkova koji će činiti najbolju putanju. Promenljiva *trencena* predstavlja cenu od inicijalnog čvora do tekućeg radnog čvora. Promenljiva C*ene* odnosi se na niz cena putanja od inicijalnog čvora do susednog čvora tekućeg radnog čvora i od inicijalnog čvora do susednog čvora prethodnog radnog čvora. Niz *preth* je niz tekućih i prethodnih radnih čvorova, a niz *susedi* je niz susednih čvorova tekućeg radnog čvora i susednih čvorova prethodnih radnih čvorova. Sada kada smo definisali parametre definišemo spoljašnju petlju koja će se izvršavati dok dužina niza *CvoroviUStablu* ne postane jednaka dužini niza *lines* tj. sve 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 *lines* 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()*** vraca -1, a u suprotnom indeks elementa *q* u datom nizu. Ako je *q* već ukljucen 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:

```
var min = cene[0];
        b=0;
        var i=1;

        while (i<cene.length){
            console.log("dok je ",i,"manje od",cene.length);
            if(min>cene[i]){
                console.log("ako je",min," vece od",cene[i]);
                b=i;
```

```
                console.log("varijabla B je: ",b);
            }
            i++;
        }
```

Promenljivom *min* je označena najmanja cena iz niza *cene*, a promenljivom *b* indeks elementa *min*. Indeksom *b* određeni su početni i krajnji čvor linka koga treba uključiti u stablo, pa se u for ciklusu prolazi kroz sve članove niza *lines*, 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[b]*, *RadniCvor* postaje *susedi[b]*, a cena putanje do novog radnog čvora je *min*:

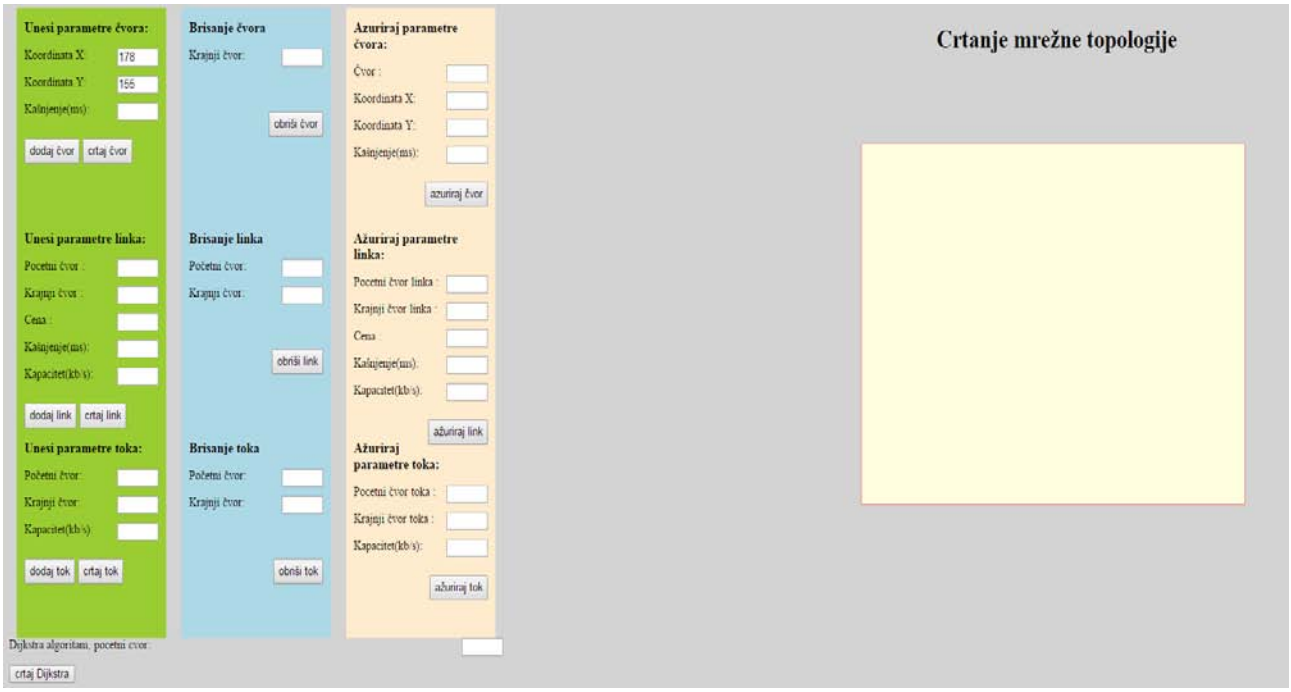
```
cvoroviUStablu.push(susedi[b]);
        console.log("cvorovi u stablu niz: ",cvoroviUStablu);
        trencena = min;
        radniCvor = susedi[b];
```

Kako u nizu susedi može postojati više vrednosti *susedi[b]*, 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[b]*. Elemente sa tim indeksima treba ukloniti iz nizova *susedi*, *preth* i *cene*, što je i urađeno u sledećem for ciklusu:

```
for(

var v=susedi.length-1; v >= 0; v++){
            if(susedi_out.indexOf(v) >= 0){
                susedi.splice(v,1);
                cene.splice(v,1);
                preth.splice(v,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. I na kraju dve funkcije *drawNodes* i *drawLines* za iscrtavanje rešenja Dijkstra algoritma.



Sl.5.2.1. - Prikaz veb stranice sa Dijkstra algoritmom

6. ZAKLJUČAK

HTML5 jezik nudi brojne mogućnosti koje programerima veb aplikacija i stranica omogućavaju jednostavnu realizaciju složenih funkcionalnosti. Veliki pomak HTML5 nudi u oblasti grafičkog prikaza uvođenjem canvas i svg elemenata. Svg element je veoma pogodan za crtanje vektorske grafike tj. pravilnih geometrijskih oblika. Samim tim svg element je izuzetno pogodan za iscrtavanje mrežne topologije u okviru veb stranice.

U okviru teze je realizovana biblioteka koja omogućava da se na jednostavan način iscrta mrežna topologija na veb stranici pomoću svg elementa koji je najadekvatniji za ovu primenu. Realizovana biblioteka je detaljno opisana i omogućava kvalitetno definisanje čvorova, linkova i tokova u mreži. Korisnik na lak način može da integriše razvijeni javascript fajl tj. biblioteku u svoj veb sajt i da ga iskoristi za demonstraciju i edukaciju. U tezi je dat jedan primere primene biblioteke, ali mogućnosti su brojne.

LITERATURA

[1] W3C, http://www.W3C.org

[2] HTML5 differences HTML4, http://dev.w3.org/html5/html4-differences/

[3] HTML5 differences HTML4, http://www.w3.org/TR/2011/WD-html5-diff-20110405/

[4] HTML5-SVG, http://www.tutorialspoint.com/html5/html5_svg.htm

[5] SVG Tutorial, https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial

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