SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij računarstva

VIZUALIZACIJA STRUKTURE GRAF

Završni rad

Dario Ćorić

Osijek, 2018.

SADRŽAJ

1. UVOD

Graf je struktura podataka koju čine čvorovi i grane, pri čemu su čvorovi međusobno povezani granama. Grafovi imaju široku primjenu u mnogim granama znanosti gdje je potrebno prikazati međusobnu povezanost podataka.

U ovom završnom radu opisan je postupak stvaranja web aplikacije za prikaz grafa. Opisane su tehnologije korištene za razne funkcionalnosti aplikacije. Dio aplikacije vidljiv korisniku realiziran je pomoću jezika HTML, CSS i JavaScript te biblioteke vis.js, dok je pokretanje poslužitelja omogućeno koristeći program izvršenja (*run-time environment*) Node.js i više biblioteka i okvira. Opisan je postupak kreiranja samog projekta i uključivanja spomenutih biblioteka i okvira u njega. U zaključku je napravljen osvrt na rad i dotaknute su mogućnosti proširenja aplikacije.

1.1. Zadatak završnog rada

Potrebno je napraviti web aplikaciju za vizualizaciju strukture podataka graf. Aplikacija mora imati mogućnost zadavanja grafa, zapisa zadanog grafa u datoteku te čitanja i crtanja grafa iz datoteke.

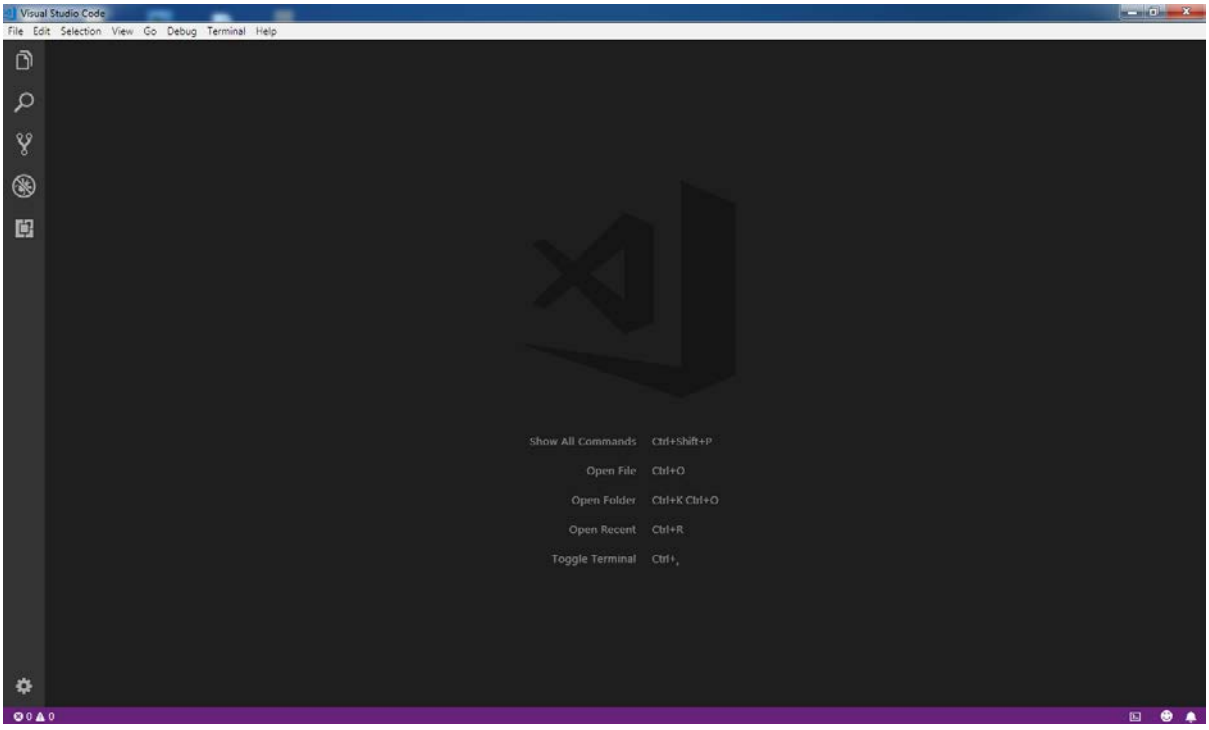
2. TEHNOLOGIJE KORIŠTENE U RADU

Kako bi bilo moguće napraviti programsko rješenje, potrebno je razvojno okruženje i dodatne tehnologije za razne funkciolnosti. Kao razvojno okruženje korišten je Visual Studio Code i aplikacija je pisana većinom u JavaScript-u uz osnovne elemente HTML-a i CSS-a. Za programiranje *servera* (poslužitelja) korišten je programski okvir (eng. *framework*) Express.js a izvršava se sustavom node.js. Korištena su i sljedeće biblioteke: node.js File System, vis.js, body-parser, underscore, uuid. Svrha svake od njih bit će objašnjena u nastavku rada.

2.1. Visual Studio Code

Visual Studio Code je program za uređivanje teksta razvijen od strane Microsoft-a za operacijske sustave Windows, Linux i macOS. Osnova mu je na programski okvir Electron. U programu je omogućeno dodavanje prevoditelja za brojne jezike (C, C++, JavaScript, C# itd.).

Prilikom stvaranja projekta okruženje ima mogućnost učitati direktorij i postaviti ga kao korjenski direktorij projekta. Nakon toga korisnik dobiva mogućnost stvaranja različitih datoteka unutar direktorija (JavaScript, HTML, JSON...) desnim klikom na direktorij u sučelju. Okruženju se na jednostavan način mogu biti dodana proširenja kao npr. terminal za naredbe iz kojeg je moguće testirati projekt i ispisivati poruke.



Slika 2.1. *Početni prozor Visual Studio Code-a*

U anketi provedenoj među korisnicima stranice *Stack Overflow* 2018. godine pokazano je da je VS Code najčešće korišteno okruženje, korišteno od strane 34.9% ispitanika.

2.2. JavaScript i Node.js

JavaScript [1] (često skraćeno JS) je objektno-orjentirani programski jezik višestruke paradigme. Prvi put se pojavljuje 1995. godine. Uz HTML i CSS jedan je od ključnih jezika u razvoju web aplikacija. Svi značajniji internet preglednici (Chrome, Firefox, IE/MS Edge) imaju ugrađen prevoditelj za JS kod (eng. *interpreter*). Iako je u početku korišten samo za programiranje korisničke strane *web* stranica, danas je korišten za programiranje poslužiteljskog dijela *web* stranica i baza podataka.

Node.js [2] je otvoreni (eng. *open-source*) višeplatformni (eng. *cross-platform*) sustav izvršenja (eng. *run-time environment*) koji izvršava JavaScript kod izvan preglednika. Njime je programerima omogućeno pisanje serverskih skripti (eng. *server-side scripting*) kojima se stvara dinamični sadržaj prije slanja sadržaja pregledniku. Arhitektura sustava je zasnovana na događajima (eng. *event-driven architecture*) i ima mogućnost asinkronog ulaza i izlaza. Nakon što je instaliran na računalu, omogućuje dodavanje biblioteka (eng. *library*) u projekt koristeći konzolnu naredbu *npm install (biblioteka)* te pokretanje aplikacije naredbom *node (glavna datoteka)*.

Uz node koristi se okvir Express kojemu je uloga stvaranje servera i rukovanje zahtjevima koji se šalju na određene putanje.

2.3. jQuery

jQuery je otvorena JavaScript biblioteka stvorena za lakše programiranje korisničke strane (*client-side scripting*). Sintaksa omogućuje lakši pristup DOM elementima (npr. gumb), upravljanje događajima (npr. klik na gumb) i stvaranje Ajax aplikacija [3]. U radu je korišten za pozivanje Express metoda pri uređivanju grafa od strane korisnika.

2.4. JSON

JSON (JavaScript Object Notation) je format zapisa za spremanje i učitavanje podataka. Osnova mu je sintaksa JavaScript objekata. Razlika u odnosu na nju je ta što se nazivi objekata u JSON-u pišu u navodnim znacima.

U nastavku se nalazi primjer JSON zapisa:

```
{
   "nodes":[
      {
         "id":"1",
         "label":"1"
      },
      {
         "id":"2",
         "label":"2"
      },
      {
         "id":"3",
         "label":"3"
      }
   ],
   "edges":[
      {
         "from":"1",
         "to":"2",
      },
      {
         "from":"2",
         "to":"3",
      },
      {
         "from":"3",
         "to":"1",
      },
      {
         "from":"1",
         "to":"1",
      }
   ],
   "options":{}}
```

Primjer 2.1. Primjer JSON zapisa

3. STVARANJE PROJEKTA I FRONT-END RJEŠENJE

3.1. Stvaranje i inicijacija projekta

U VS Code-u omogućen je uvoz direktorija i određivanje istog korjenskim direktorijem projekta. U ovom radu stvoren je direktorij *graphVis* i dodan je u sučelje. U korjenski direktorij dodana je datoteka *index.js* koja će sadržavati glavninu koda. Također je stvoren direktorij *public* koji sadrži datoteku *index.html* u kojoj će biti definirano sučelje za prikaz grafa i dio koda koji se pokreće iz *front-end*-a.

U terminal sučelja unešena je naredba *npm init* koja inicira projekt. Pozivanjem te naredbe dobijeni su upiti kojima se od strane sustava zahtijevaju naziv projekta, verzija, opis, glavna datoteka, testna naredba, git repozitorij, ključne riječi, autor i licenca. Ti podatci bivaju spremljeni u datoteku *package.json* koja se sprema u korjenski direktorij.

Svakim ugrađivanjem dodatne biblioteke u projekt dodana biblioteka bit će spremljena u popis *dependencies* u toj datoteci. To se postiže naredbom *npm install --save (biblioteka).* Tom listom će se nekome tko eventualno bude radio s tim projektom pokazati što je sve potrebno za razvoj.

3.2. HTML i vis.js

U datoteci *index.html* stvoren je okvir koji će biti podloga za crtanje grafa. Definiran je kao *div* element iz razloga što je u biblioteci vis.js omogućena manipulacija takvim elementom. U *head* elementu referencirane su dvije potrebne biblioteke i jedna CSS datoteka: *vis.js*, *jQuery.js* te *vis.css*.

Funkcijama iz biblioteke *vis.js* moguće je stvaranje i uređivanje čvorova i grana grafa dok je datotekom *vis.css* definiran njihov osnovni izgled [4]. Pri uobičajenim opcijama čvorovi su prikazani kao plavi krugovi ili elipse no to je moguće promijeniti. Također je moguće grupirati podatke po svojstvima za bolji, pregledniji prikaz. Biblikoteka posjeduje ugrađene funkcije za dinamično uređivanje čvorova i grana no ograničenje joj je to da se u tim funkcijama ti podaci nigdje ne spremaju. Zbog toga ih je potrebno proširiti, što je opisano u idućem poglavlju rada.

```
<div id ="graph"></div>
    <br>
    <div id = "download"></div>
    <br>
    <div>
        <input type="file" id="fileToLoad">
        <br>
        <button onclick="loadFileAsText()">Učitaj odabranu datoteku</button>
</div>
```

Isječak 3.1. HTML korisničkog sučelja

Gornjim HTML-om dobivamo sljedeće:

Slika 3.1. Izgled aplikacije

Bibliotekom *jQuery* se dobiva pristup *back-end* dijelu rješenja iz *front-end*-a. Korištena je za definiranje funkcija za dodavanje, uređivanje i brisanje elemenata ugrađenih u *vis.js* na način da se pozovu funkcije koje mogu izvršiti i spremiti promjene na zadanu destinaciju na localhost-u (*localhost/api*).

U sljedećem dijelu koda određena je površina za crtanje grafa i dodatne mogućnosti koje daje vis:

```
$.get(„/api“, function(data){
            data.options.manipulation = {
                addNode: addNode,
                editNode: editNode,
                deleteNode: deleteNode,
                addEdge: addEdge,
                editEdge: editEdge,
                editEdge: editEdge
            }
            var container = document.getElementById(„graph“);
            var network = new vis.Network(container, data, data.options);

 var dat = „text/json;charset=utf-8,“ + encodeURIComponent(JSON.stringify(data));

            var a = document.createElement('a');
            a.href = 'data:' + dat;
            a.download = 'data.json';
            a.innerHTML = 'Preuzmi podatke o grafu';

            var download = document.getElementById('download');
            download.appendChild(a);
        });
```

Isječak 3.2. Metode vis.js-a, crtanje i preuzimanje grafa

Kod se izvršava učitavanjem početne stranice (*127.0.0.1/8080/*). Šalje se zahtjev za sadržaj na adresi */api* (JSON zapis grafa) za što je definirana metoda u back-endu. U objektu data.options.manipulation pridruženi su pozivi ostalim metodama za rad nad podatcima u grafu sa ugrađenim gumbovima.

Nadalje, varijabli *container* pridružuje se element na kojem će se prikazati graf. Zatim se definira varijabla *network* kojom se vrši prikaz pomoću *vis* metode *Network*.

Tu je također definiran i element za preuzimanje podataka o stvorenom grafu (*a*). Kao sadržaj pridružen mu je kodirani oblik JSON zapisa strukture grafa koji klikom biva zapisan u datoteku u JSON obliku.

Učitavanje grafa iz datoteke omogućeno je sljedećim dijelom koda:

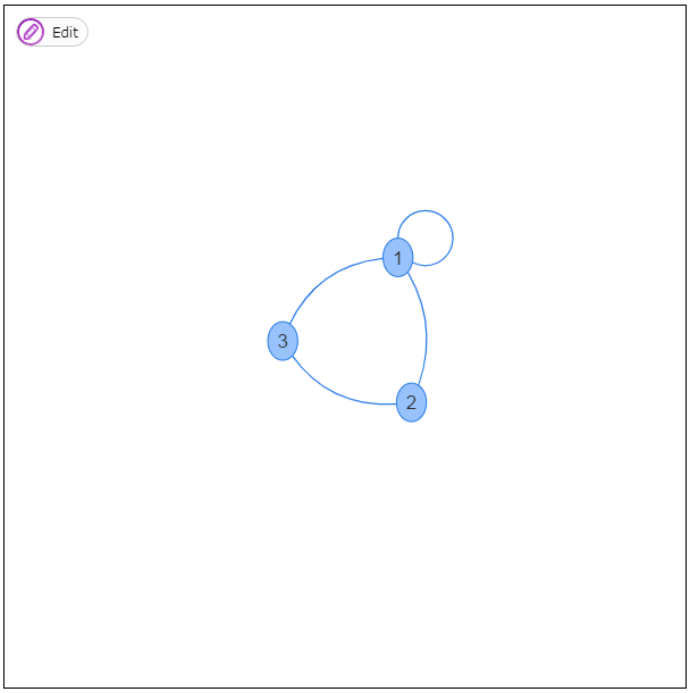
```
function loadFileAsText(){
                var fileToLoad = document.getElementById(„fileToLoad“).files[0];

                var fileReader = new FileReader();
                fileReader.onload = function(fileLoadedEvent){
                    var textFromFileLoaded = fileLoadedEvent.target.result;
                    var data = JSON.parse(textFromFileLoaded);
                    var container = document.getElementById(„graph“);
                    var network = new vis.Network(container, data, data.options);
                };

                fileReader.readAsText(fileToLoad, „UTF-8“);
            }
```

Isječak 3.3. Učitavanje grafa iz datoteke

Nedostatak ovakvog učitavanja iz datoteke je taj što podatci o grafu neće biti sačuvani na poslužitelju te će prvim osvježavanjem aplikacije biti izbrisani i graf će nestati.



Slika 3.2. Graf učitan iz datoteke

Zapis grafa na slici 3.2. je identičan Primjeru 2.1. (str. 4).

Klikom na gumb *Edit* u gornjem lijevom kutu okvira otvaraju se gumbovi *Add Node* i *Add Edge* ukoliko nijedan element postojećeg grafa nije označen. To je prikazano na sljedećoj slici:

Slika 3.3. Gumbovi za dodavanje čvora i grane

Klikom na gumb *Add Node* i na mjesto gdje ga želi dodati korisnik dobiva upit za unos id i label parametara čvora. Ako je ijedno od polja prazno, čvor neće biti dodan. Analogno, klikom na *Add Edge* korisnik može povući granu između dva čvora *drag-and-drop* metodom ili stvoriti granu na jednom čvoru klikom na njega.

Ukoliko je odabran čvor u grafu, bit će prikazani gumbovi na slici:

Slika 3.4. Gumbovi za uređivanje i brisanje čvora

Uređivanje čvora podrazumijeva samo promjenu oznake koja je prikazana u vizualizaciji. Korisnik ne može ovim putem mijenjati nijedno drugo svojstvo čvora. Brisanjem čvora bit će obrisane sve grane koje izviru iz čvora.

Analogno, ukoliko je odabrana grana grafa, bit će prikazani gumbovi za uređivanje i brisanje grane. Uređivanje podrazumijeva povlačenje na drugi čvor. Brisanjem grane čvorovi iz kojih je izvirala ostaju netaknuti.

4. BACK-END RJEŠENJE

Back-end dio rješenja potreban je za zapisivanje promjena u strukturi grafa.

4.1. Lokalni server uz Express.js

Za stvaranje lokalnog servera u projektu je korišten *framework* Express.js. Da bi bila omogućena njegova uporaba, u terminal upisujemo naredbu *npm install –save express* kojom se u projekt dodaju datoteke potrebne za rad *framework*-a. Nakon toga se koristi funkcija *require(„express“)* kojom se varijabli *express* predaju sve funkcionalnosti *framework*-a, nakon čega se varijabli *app* predaje *express()*. Time je omogućen pristup metodama Express-a (*get*, *put*, *post*, *delete*, *listen*) putem te varijable. Na sličan način uključene su i ostale biblioteke:

```
var express = require("express");
var bodyParser = require("body-parser"); //omogucuje parsiranje html elemenata
var _ = require("underscore"); //omogucuje prolaz kroz niz cvorova i bridova
var uuid = require("uuid"); //omogucuje zadavanje uuid-a
var fs = require('fs'); //omogucuje citanje i pisanje datoteka
var JsonDB = require('node-json-db'); //omogucuje direktrno citanje i pisanje
                                      //json datoteke

var app = express();

app.use(express.static('public'));
app.use(express.static('node_modules'));

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());
```

Isječak 4.1. Inicijacija biblioteka i okvira

Koristeći metodu *listen* stvoren je lokalni server na port-u 8080:

```
app.listen(8080, function() {
    console.log("Server started on port 8080.");
});
```

Isječak 4.2. „Slušanje“ na portu 8080

Time je u pregledniku omogućen pristup aplikaciji na adresi *127.0.0.1:8080/*.

4.2. Inicijacija grafa i metode nad grafom

Graf je u kodu definiran kao polje praznih objekata spremljeno u varijablu *data*. Objekti su *nodes* (čvorovi), *edges* (grane) i *options*. Podatci se vraćaju svakim učitavanjem stranice:

```
app.use("/api", function(req,res){
    res.send(data).end(); });
```

Isječak 4.3.

Koristeći Express *routing* metode moguće je dinamično mijenjati zapis elemenata grafa na serveru. Kao argument metode primaju putanju na koju je poslan zahtjev a vraćaju rezultat funkcije koja kao argument prima tijelo zahtjeva i vraća odgovor. Korištene su metode *post*, *put* i *delete*.

```
app.post("/api/node", function(req, res){//metoda za dodavanje cvora
    var newNode = req.body;//predaje cvor pomocu bodyparser-a
    var foundNode = _.find(data.nodes, function(node) {
        return node.id === newNode.id;
    });
    if(foundNode) {        //provjerava postoji li cvor s istim id-om
        return;
    }

    data.nodes.push(newNode);//sprema cvor u data

    res.send(newNode).end();//vraca cvor u HTML} );
```

Isječak 4.4. Dodavanje čvora na server

Za dodavanje novog čvora potrebno je najprije provjeriti postoji li već čvor s istim id-om. U tome pomaže biblioteka *underscore* koja sadrži ugrađene metode za rad s nizovima. Ukoliko nije pronađen isti čvor, novi čvor se dodaje u objekt *data* i zapisuje se na */api*. U suprotnom metoda vraća *null* i samim time čvor nije dodan.

```
app.put("/api/node", function(req,res) {//uredjivanje cvora
    var updatedNode = req.body;
    var realizedNode;
    _.each(data.nodes, function(node) {
        if(node.id === updatedNode.id) {
            node.label = updatedNode.label;
            realizedNode = node;
        }
    });
      res.send(realizedNode).end();} );
```

Isječak 4.5. Uređivanje čvora

Metoda za uređivanje čvora također koristi *underscore* metodu koja pronalazi traženi čvor u nizu i mijenja mu oznaku (*label*). Pronađeni čvor se sprema u *realizedNode* i vraća kao odgovor.

```
app.delete(„/api/node“, function(req,res) {//brisanje cvora
    var deletedNode = req.body.node;
    var deleteResult = {//spremnik za cvorove i grane koji
        nodes: [],      //nece biti obrisani
        edges: [] }
      var updatedNodes = _.filter(data.nodes, function(node) {//pronalazi cvor
za brisanje
        var keep = (node.id !== deletedNode);
        if(!keep) {
            deleteResult.nodes.push(node);
        }
        return keep;
    });
    var updatedEdges = _.filter(data.edges, function(edge) {//pronalazi grane za
brisanje
        var keep = (edge.from !== deletedNode) && (edge.to !== deletedNode);
        if(!keep) {
            deleteResult.edges.push(edge);//brise grane na brisanom cvoru
        }
        return keep;
    });
    data.nodes = updatedNodes;//sprema preostale cvorove u data
    data.edges = updatedEdges;

    res.send(deleteResult).end();
});
```

Isječak 4.6. Brisanje čvora i susjednih grana sa servera

Brisanje čvora ima više koraka. Ako su iz njega izvirale grane, potrebno je i njih obrisati. Iniciran je objekt *deleteResult* u kojem će biti sadržani čvor i grane. Metodom *filter* najprije se pronalazi čvor koji se briše a ostali čvorovi se spremaju u privremenu varijablu. Na sličan način pronalaze se grane koje će biti obrisane. Nakon toga se ažurira objekt *data*.

```
app.post("/api/edge", function(req,res) {//dodavanje grane
```

```
    var newEdge = req.body;
    newEdge.id = uuid();//daje dinamicni uuid grani
    data.edges.push(newEdge);

    res.send(newEdge).end();
});
```

Isječak 4.7. Dodavanje grane u server

Dodavanje grane vrši se na skoro identičan način kao i dodavanje grafa. Metoda se razlikuje u tome što se grani svojstvo *id* ne generira automatski zbog čega je potrebno pozvati funkciju *uuid* koja generira nasumični 128-bitni identitet.

```
app.put("/api/edge", function(req,res) {//uredjivanje grane
    var updatedEdge = req.body;
    var realizedEdge;
    _.each(data.edges, function(edge) {
        if (edge.id === updatedEdge.id) {
            edge.from = updatedEdge.from;//mijenja pocetni ili krajnji
            edge.to = updatedEdge.to;    //cvor grane
            realizedEdge = edge;
        }
    });
    res.send(realizedEdge).end();});
```

Isječak 4.8. Uređivanje grane

Uređivanje grane je, kao i stvaranje, slično uređivanju čvora. Nakon što je odabrana grana pronađena, ažurira se početni ili krajni čvor grane budući da se svakim pozivom može promijeniti samo jedno od to dvoje.

```
app.delete("/api/edge", function(req, res) {//brisanje grane
    var deletedEdge = req.body.edge;//
    var deleteResult = {//spremnik za granu koja ce biti obrisana
        nodes: [],
        edges: [] }
    var updatedEdges = _.filter(data.edges, function(edge) {
        var keep = (edge.id !== deletedEdge
        if(!keep) {
            deleteResult.edges.push(edge);
        }
        return keep;
    });
    data.edges = updatedEdges;//sprema preostale grane u data
    res.send(deleteResult).end();});
```

Isječak 4.9. Brisanje grane sa servera

Brisanje grane prati sličan uzorak. Razlikuje se od brisanja čvora po tome što ova metoda ne briše susjedne čvorove obrisane grane.

4.3. jQuery/Ajax zahtjevi

Pozivi Express metoda tj. zahtjevi na određene putanje realizirani su pomoću jQuery i Ajax zahtjeva. Kad korisnik napravi promjenu na grafu, poziva se jedna od sljedećih funkcija ovisno od vrste promjene.

```
var addNode = function(data, callback) {
    var id = prompt("Unesite id:");
    if(!id) {return;}
    var label = prompt("Unesite oznaku:");
    if(!label) {return;}

    data.id = id;
    data.label = label;
    $.post("api/node", data, function(result){
        callback(result);
    });
};
```

Isječak 4.10. Definiranje čvora i poziv za dodavanje

Pri dodavanju čvora korisnik naprije dobiva upit za unos id-a i oznake čvora. Ukoliko ništa nije uneseno za jednu ili obe vrijednosti, čvor neće biti dodan. Podaci se spremaju u varijablu data koja se predaje kao argument pri *post* zahtjevu na adresu */api/node*.

```
var editNode = function(data, callback) {
    var label = prompt("Unesite oznaku:", data.label);
    data.label = label;

    $.ajax({
        method: "put",
        url: "/api/node",
        data: data,
        success: function(result) {
            callback(result);
        }
```

```
        });
    };
```

Isječak 4.11. Definiranje promjene čvora i poziv promjene

Kod uređivanja čvora korisnik dobiva mogućnost unosa nove oznake. Budući da jQuery nema definiranu metodu za *put* zahtjev, potrebno ju je definirati pomoću *ajax* metode. Kao opcije metode potrebno je zadati *method* (vrsta HTTP zahtjeva), putanju na koju će se zahtjev napraviti (*url*), podatke koji se predaju kao argument (*data*) te se određuje funkcija koja će se izvršiti u slučaju uspješnog zahtjeva (*success*). Ta funkcija vraća obrađene podatke iz ranije definiranih Express metoda.

```
    var deleteNode = function(data, callback) {
        $.ajax({
            method: "delete",
            url: "/api/node",
            data: {node: data.nodes[0]},
            success: function(result) {
                callback(result);
            }
        });
    };
```

Isječak 4.11. Poziv za brisanje odabranog čvora

Također je potrebno pomoću Ajaxa definirati metodu za slanje *delete* zahtjeva pri brisanju čvora.

Stvaranje, uređivanje i brisanje grana izvedeno je na isti način kao i za čvorove:

```
    var addEdge = function(data, callback) {
        $.post("api/edge", data, function(result){
            callback(result);
        })
    };
```

Isječak 4.12. Poziv za dodavanje nove grane

```
var editEdge = function(data, callback) {
      $.ajax({
          method: "put",
          url: "/api/edge",
          data: data,
          success: function(result) {
              callback(result);
          }
      })
  };
```

Isječak 4.13. Poziv za uređivanje grane

```
  var deleteEdge = function(data, callback) {
      $.ajax({
          method: "delete",
          url: "/api/edge",
          data: {edge: data.edges[0]},
          success: function(result) {
              callback(result);
          }
      });
  };
```

Isječak 4.14. Poziv za brisanje grane

Koristeći ove metode u kombinaciji sa Express metodama omogućuje zapis podataka na server. Oni ostaju zapisani sve dok je server aktivan ili dok ne dođe do nove promjene.

5. ZAKLJUČAK

Cilj ovog rada je bio stvaranje web aplikacije za unos grafa i rad s njime. Ona omogućuje unos čvorova i granja te njihovo uređivanje i brisanje koristeći sučelje. Također omogućuje spremanje zadanog grafa u tekstualnu datoteku i uvoz grafa iz tekstualne datoteke. Rješenje izvedeno je pomoću HTML-a, CSS-a, JavaScript-a, Node.js-a i proširenja od kojih su najznačajnija Express, jQuery i vis.js. Pomoću HTML-a i CSS-a stvoreno je sučelje za unos, uvoz i izvoz grafa a korisnički rad s njime omogućen je JavaScript-om i jQuery-jem. Sama vizualizacija omogućena je bibliotekom vis.js. Poslužitelj je stvoren koristeći Express okvir Node.js-a. Aplikacija bi se mogla unaprijediti dodavanjem mogućnosti grupiranja podataka npr. po boji ili obliku čvora ili spremanjem tekstualnih datoteka na poslužitelj.

Aplikaciju je moguće proširiti u više pogleda. Trenutno je ograničena na prikazivanje neusmjerenih grafova no moguće je podesiti *vis* za prikaz usmjerenih grafova. Još neke mogućnosti proširenja su trajno spremanje JSON zapisa u bazu podataka radi lakšeg pristupa i određivanje oblika čvora ovisno od parametara.

LITERATURA

[1] w3schools.com (1998.): https://www.w3schools.com/js/ (15.9.2018.

[2] nodejs..org (2009.): https://nodejs.org/en/docs/guides/ (15.9.2018.)

[3] jquery.com (2006.): https://api.jquery.com/ (15.9.2018.)

[4] visjs.org, (2015.): http://visjs.org/docs/network/ (15.9.2018.)

SAŽETAK

Cilj rada bio je opisati stvaranje web aplikacije za vizualizaciju grafa. Opisani su stvaranje poslužitelja za aplikaciju i metode za rad s podatcima o grafu. Moguće je stvoriti graf pomoću sučelja ili čitanjem iz tekstualne datoteke. Korištene su tehnologije JavaScript, Node.js, Express i biblioteka vis.js, uz još neke pomoćne biblioteke.

ključne riječi: graf, javascript, jquery, node.js, čvor, grana, json, vis.js

ABSTRACT

Graph Structure Visualization

The aim of this paper was to describe the creation of a graph visualization web application. The creation of a server and methods for graph data manipulation were described. A graph can be created using the application interface or by reading from a text file. The technologies used are JavaScript, Node.js, Express and the vis.js library, along with other libraries.

keywords: graph, javascript, jquery, node.js, node, edge, json, vis.js

ŽIVOTOPIS

Dario Ćorić rođen je 1.3.1996. u Dubrovniku. Po završetku osnovne škole 2010. godine upisuje Opću gimnaziju Katoličkog školskog centra „Sv. Pavao“ u Zenici koju završava 2014. godine. Iste godine upisuje sveučilišni preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.