UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LearnGraphs

O aplicație didactică asupra teoriei grafurilor

Damian Răzvan Codrin

Sesiunea: iulie, 2017

Coordonator științific

Lector, Dr. Ionuț Pistol

Cuprins

1 Introducere 3
2 Aplicații similare și tehnologii folosite în acestea 6
2.1 Graph Creator 6
2.2 Graph Online 8
2.3 GoJS 10
2.4 Alchemy.js 11
2.5 Cytoscape.js 13
2.6 Concluzii 13
3 Tehnologii folosite 14
3.1 Angular 2 14
3.2 Vis.js 15
3.2.1 Module 17
3.2.2 Metode 21
3.2.3 Evenimente 23
3.3 ng-sidebar 24
3.4 Concluzii 26
4 Descrierea aplicației LearnGraphs 27
4.1 Analiză și proiectare 27
4.2 Implementare 30
4.2.1 Routing 30
4.2.2 Editarea grafului 32
4.2.3 Desenarea grafului 32
4.2.4 Implementarea algoritmilor 33

4.3 Manual de utilizare 34
4.3.1 Adăugarea nodurilor și a muchiilor 34
4.3.2 Exportarea și importarea grafurilor 37
4.3.3 Alegerea setărilor generale pentru graf 38
4.3.4 Editarea nodurilor și a muchiilor 39
4.3.5 Rularea unor algoritmi pe grafurile create sau importate 41
5 Concluzii 42
Bibliografie 43
Anexa 1 45

Capitolul 1

Introducere

Pentru această lucrare de licență m-am documentat cu privință la teoria grafurilor și diferite aspecte legate de această temă atât din cursurile de „Algoritmica Grafurilor”, parcurse de-a lungul semestrului întâi al anului doi de facultate, cât și cele de „Inteligență Artificială” din anul 3, aici interesându-mă în principal de rezolvarea problemelor de IA cu ajutorul grafurilor (alte surse: [1], [2]). Ca urmare a acestor studii am lucrat la crearea unei aplicații cu ajutorul căreia se pot crea grafuri cu ușurință, având controale accesibile și ușor de utilizat. Însă crearea lor nu este singura funcționalitate a acesteia, ci se pot aplica algoritmi specifici pentru grafuri. Pentru crearea aplicației am studiat unul dintre cele mai cunoscute și folosite framework-uri JavaScript la ora actuală, și anume Angular 2. În ceea ce privește crearea grafurilor, m-am documentat despre o serie de biblioteci JavaScript, în cele din urmă folosind-o pe cea care oferea cele mai multe posibilități.

Una dintre metodele de reprezentare vizuală a problemelor și cea despre care este vorba în aceasta lucrare este cea sub formă de grafuri. Un graf este o diagramă formată din puncte, numite noduri, unite între ele prin linii, numite muchii, fiecare dintre aceasta unind două noduri.

Grafurile sunt folosite în numeroase domenii ori de câte ori este nevoie de o reprezentare logică a legăturii dintre obiecte. Unul dintre cele mai relevante exemple în care grafurile sunt folosite este aplicația fără de care viața șoferilor ar fi cu mult mai grea, cel puțin atunci când indicatoarele rutiere nu prea le sunt de folos spre a ajunge la destinație. GPS-ul, acronimul din limba engleză de la Global Positioning System, este folosit de numeroase aplicații care au scopul de a servi utilizatorului ca unealtă pentru înștiințarea sa în legătură cu drumul de urmat pentru atingerea punctului de destinație a șoferului (Despre GPS - [3], Cele mai bune 10 aplicații android ce folosesc GPS – [4]).

În acest exemplu, obiectele sau nodurile sunt orașele sau alte puncte de interes, iar ce le leagă pe toate acestea sunt șoselele (muchiile) prin intermediul cărora șoferul se poate deplasa dintr-un oraș într-altul. De aici este alegerea utilizatorului în legătură cu traseul de urmat. Poate

alege traseul cel mai scurt, însă, în orele de vârf, timpul în care ar ajunge la destinație s-ar putea sa fie mai mic pe un traseu mai lung pus la dispoziție de aplicație.

O altă formă de utilizare a grafurilor este reprezentată de diagramele UML (Unified Modeling Language). În unele situații nodurile pot reprezenta acțiuni, stări, actori ce iau parte la acțiune, situații ș.a.m.d. Muchiile pot fi asocieri, moșteniri de clase, dependențe, condiții, tranziții ș.a.m.d. Acestea, puse împreună, formează o reprezentare grafică folosită pentru descrierea de modele în ingineria programării și vizualizarea design-ului unui sistem (Tipuri de diagrame UML – [5], Program special de desenare diagrame UML ArgoUML - [6]).

Scopul acestei lucrări de licență este de a crea o aplicație care va fi folosita în cadrul didactic pentru a facilita munca în scopul predării celor două mari capitole din timpul liceului dedicate grafurilor și arborilor.

Aplicația ajută cadrul didactic în predarea acestor capitole deoarece conțin unele din cele mai complexe noțiuni, astfel facilitând atât treaba profesorilor cât și cea a elevilor. Cu ajutorul acesteia, profesorii vor putea crea grafuri sau arbori conform lecției sau, cel mai important, conform oricărei probleme propusă de dascăl. În plus față de asta, se va putea de asemenea încarcă un fișier care conține toate datele creării unui graf, date scrise într-un format adus la cunoștința utilizatorului aplicației. Astfel, acesta își va putea pregăti dinainte un exemplu relevant pentru elevi, nefiind nevoit sa stea sa-l creeze pe loc, important fiind exemplul în sine, și nu crearea acestuia.

O altă funcționalitate, de care ar dispune, este crearea unui graf, cu noduri și muchii, având stilizarea dorită pentru fiecare nod sau muchie în parte, ca mai apoi, acesta putând fi salvat local tot sub forma unui fișier cu formatul menționat mai sus. Toate acestea fiind făcute, graful nostru creat și salvat va putea fi urcat din nou în aplicația noastră, având posibilitatea modificării lui din punctul în care am rămas.

Utilizatorul, odată ce și-a creat propriul graf sau arbore, va putea aplica o serie de algoritmi de parcurgere pentru acesta, ilustrând elevilor într-un mod vizual și pas cu pas, pentru a nu rata anumite concepte importante, mecanismul traversării.

Teoria grafurilor se studiază în diverse universități fiind considerată o materie foarte importantă în informatică. Spre exemplu, cei de la Universitatea Southampton (din Marea Britanie) predau teoria grafurilor, materia fiind structurată în așa fel încât studenții interesați, în urma terminărilor acestor cursuri, să rămână cu cele mai importante aspecte [7].

În continuare capitolele acestei lucrări de licență vor fi descrise într-un mod succint pentru a ilustra conținutul acestora cititorului.

Acest capitol este destinat punerii în cunoștință de cauză a celui care citește lucrarea în legătură cu tema licenței. Aici sunt descrise alte exemple de grafuri și cum sunt ele folosite în anumite domenii. În plus, sunt prezentate diferite surse de documentație care au ajutat la implementarea și scrierea licenței.

În Capitolul 2, **„Aplicații similare și tehnologii folosite în acestea”**, după cum spune și titlul, se vorbește despre alte aplicații care au un scop asemănător cu cel al aplicației de licență. Acest lucru intră în discuție pentru a prezenta ce anume a fost implementat de alți ingineri software în aplicațiile lor și ce anume lipsește. O mare parte din ce se regăsește în creațiile lor va trebui sa conțină și aplicația noastră, însă, din ceea ce lipsește, noi vom încerca implementarea acelor funcționalități și vom prezenta calea spre îndeplinirea acestei sarcini, dar în capitolele destinate acestora. Pe lângă toate acestea, vor fi descrise o serie de biblioteci JavaScript ,și nu numai, destinate creării de grafuri. În final se va vorbi de motivele pentru care acestea nu au fost folosite în aplicația de licență cu avantajele și dezavantajele lor.

În Capitolul 3, identificat sub titlul **„Tehnologii folosite în aplicație”**, se vorbește, în sfârșit, strict de ce anume s-a folosit în aplicație. Acesta este un capitol cu mai multe detalii tehnice ce ajută la înțelegerea pașilor făcuți în scopul realizării aplicației cu toate beneficiile și riscurile asumate.

În Capitolul 4, **„Descrierea aplicației LearnGraphs”**, se prezintă toate funcționalitățile aplicației implementate. Vom vorbi despre studiile de caz ale aplicației, acestea având ca scop un grad de inovație sporit, și nu în ultimul rând despre avantajele folosirii soluției dezvoltate.

Capitolul al 5-lea, **„Concluzii”**, vom recapitula anumite subiecte importante din cuprins, descriindu-le succint, și vom vorbi despre ce s-a realizat, cu punctarea aspectelor inovatoare.

Fiecare problemă are metoda ei cea mai eficienta de rezolvare, în funcție de datele și cerințele sale. Sunt situații în care nu se știe de unde se poate porni în rezolvarea problemei. Aici intervine stilul vizual de învățare care, de cele mai multe ori, dă roade și ajută în demararea procesului de soluționare. Reprezentarea datelor unei probleme într-un mod vizual poate duce la înțelegerea ei atât într-un timp mai scurt, cât și într-o adâncime a ei cât mai profundă. Acest lucru determină alegerea celei mai eficiente metode de abordare și soluționare a problemei.

Capitolul 2

Aplicații similare și tehnologii folosite în acestea

2.1 Graph Creator

Există și aplicații asemănătoare legate de teoria, crearea și prelucrarea grafurilor în mediul online. Una dintre ele se numește „**GraphCreator”** [8]. Această aplicație folosește la bază nodul și muchia, iar, prin intermediul ei, se pot crea și explora grafuri. Se poate determina despre un graf daca este complet, să i se parcurgă căile Euler sau Hamilton și, de asemenea, se poate lucra cu arbori de cost minim.

Funcțiile principale pentru **noduri** prin care poate interacționa utilizatorul sunt adăugarea unui nod oriunde pe tablă, selectarea unui nod, afișarea tabelei cu sau fără grila, denumirea unui nod, afișarea gradului unui nod. (Fig. 2.1)

Funcțiile principale pentru **muchii** prin care poate interacționa utilizatorul sunt adăugarea unei muchii orientate sau neorientate între oricare două noduri, curbarea unei muchii, selectarea și setarea costului acesteia. (Fig. 2.2)

Funcțiile pentru **explorarea grafului** prin care poate interacționa utilizatorul sunt desenarea grafului, unealta de ajutor pentru desenarea unei căi Euler sau a uneia Hamiltoniană. (Fig. 2.3)

Acest instrument online poate fi folosit cu ușurință atunci când se dorește înțelegerea minimă a grafurilor, și anume, cum arată acesta, din ce este format (noduri și muchii) și de câte feluri pot fi muchiile (orientate sau neorientate, având cost sau fără cost). Însă, mai departe nu se poate merge deoarece instrumentul nu prea oferă suport pentru învățarea unor algoritmi ce se pot aplica pe grafuri. Butoanele de colorare a unei cai Euler sau Hamiltoniene oferite de „**GraphCreator**” pe graful creat nu sar prea mult în ajutorul unui începător în această arie. Acestea sunt mai mult niște instrumente de ghidare pentru colorarea corecta a unei căi de acest tip.

Ca și interfață grafică și accesibilitate pentru utilizatori, aplicația este, de asemenea, la un nivel de minimă înțelegere. Funcțiile pentru diferite acțiuni de aplicat componentelor grafurilor nu sunt integrate în aplicație în așa fel încât sa fie cel mai prietenos pentru cine va folosi aplicația. Spre exemplu, prezența unui buton separat de selectare a unui nod sau muchii de pe tabla de desenare face ca regulile de folosire ale acestui instrument de creare a grafurilor să fie puțin cam complicate.

Prin urmare, acest instrument, numit „**GraphCreator**”, poate fi folosită, în primă fază, pentru familiarizarea cu anumiți termeni și concepte, dar, mai târziu, va fi nevoie de mult mai mult pentru aprofundarea acestor idei.

Fig. 2.1: Controale pentru noduri în aplicația „GraphCreator”

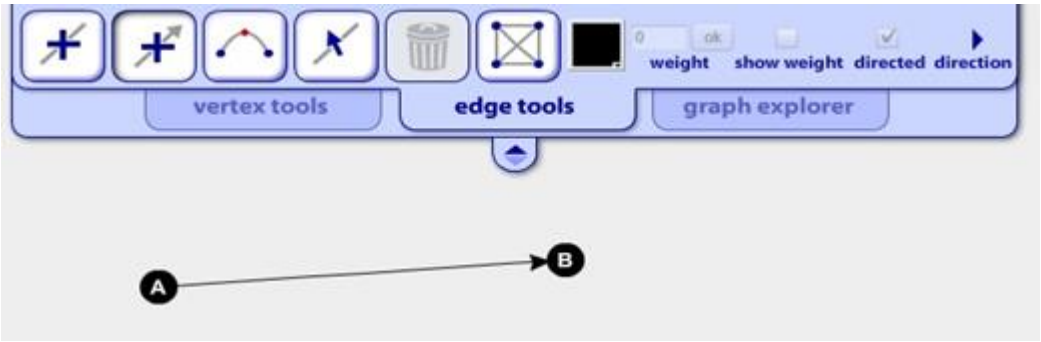

Fig. 2.2: Controale pentru muchii în aplicația „GraphCreator”

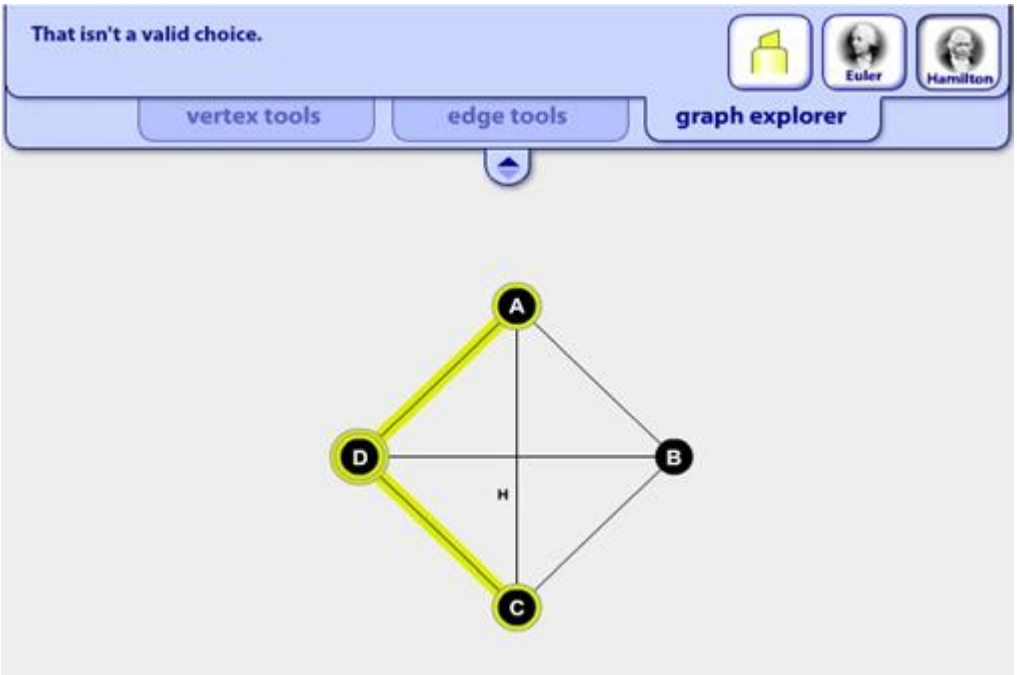

Fig. 2.3: Controale pentru explorarea grafurilor în aplicația „GraphCreator”

2.2 Graph Online

O altă aplicație destinată creării grafurilor se numește „**GraphOnline**” [9]. La fel ca aplicația precedentă crearea grafurilor constă în adăugarea de noduri pe tabla de desenare și conectarea lor prin intermediul muchiilor.

Sunt mici diferențe între cele două instrumente de desenare a grafurilor în ceea ce privește controalele pentru noduri și muchii. Primele diferențe între cele două apar la modul de creare a grafurilor. Pe lângă cel clasic, și anume, cu ajutorul cursorului, în „**GraphOnline**” se mai pot crea grafurile cu ajutorul matricelor de adiacență sau a celor de incidentă (Fig. 1.4). Acest lucru lărgește posibilitățile aplicației și reprezintă un plus față de cealaltă cea precedentă.

O altă diferență majoră este cea legată de metoda de salvare a grafurilor. De vreme ce în „**GraphCreator**” nu aveam nicio modalitate de salvare, ci doar de desenare, aici, în „**GraphOnline**”, sunt două feluri implementate, în aplicație, în care îți poți salva munca. Primul este cel sub forma de link, având posibilitatea de a fi împărtășit pe o serie de pagini de socializare puse la dispoziție. Cel de-al doilea, prin imortalizarea grafului într-o imagine ce poate fi deschisă în browser, descărcata local sau distribuită, de asemenea, pe un anumit sit de socializare. Aceste metode de salvare sunt folositoare, însă nu sunt printre cele mai eficiente (Fig. 2.4).

Cel mai mare plus pentru aplicația „**GraphOnline**” este datorat faptului că are o gamă mai largă de algoritmi dedicați grafurilor față de „**GraphCreator**” (Fig. 2.5). Astfel, această aplicație poate fi folosită atât în scopuri educaționale, formatul fiind cu mult mai accesibil, cât și în partea

de business, în cadrul unor companii atunci când este nevoie de o reprezentare a unui scenariu ori unei probleme.

Celelalte funcții și acțiuni implementate în această aplicație se aseamănă într-o anumită măsură cu precedenta. Cu toate acestea, în cele din urmă, în „**GraphOnline**” se regăsește o mai mare accesibilitate pentru utilizator. De data aceasta interfața este una mai prietenoasă pentru cei care folosesc această aplicație. Și totuși, aceasta ar putea fi mult mai accesibilă și ușor de utilizat (user-friendly).

În concluzie, cele două aplicații pentru crearea grafurilor sunt folositoare, însă, luând de la fiecare acele elemente care ușurează procesul de învățare și punându-le într-o altă aplicație, s-ar putea crea o alta cu mult mai multe elemente accesibile utilizatorului.

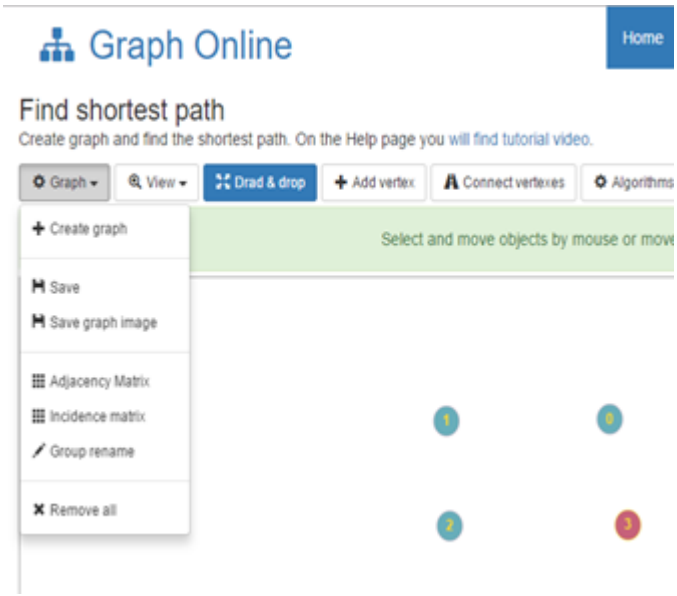

Fig. 2.4: Metode de creare și salvare a unui graf în aplicația „GraphOnline”

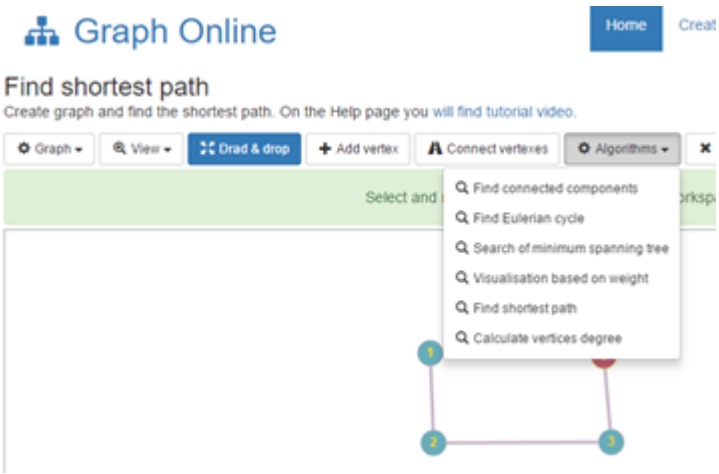

Fig. 2.5: Algoritmi dedicați grafurilor în aplicația „GraphOnline”

În continuare sunt prezentate o serie de tehnologii ce pot fi folosite în crearea de aplicații de desenare și vizualizare de grafuri. Voi aborda în descrierea lor atât aspecte tehnice, cât și detalii referitoare la experiența utilizatorului și diversitatea funcțiilor oferite.

2.3 GoJS

GoJS ([10]) este o bibliotecă JavaScript dedicată creării de diagrame și grafuri interactive. Aceasta poate rula complet în browser prin randarea într-un element Canvas HTML5 sau SVG și nu necesită prezența unui server pentru a oferi interactivitate cu utilizatorul. Utilizarea GoJS-ului nu este dependentă de folosirea unui anumit framework JavaScript, ceea ce o face să fie versatilă și ușor de adoptat în orice tip de aplicație. Popularitatea acestei biblioteci este dovedită prin numărul mare de aplicații în care este încorporată. O listă a celor mai semnificative produse ce folosesc GoJS-ul poate fi găsită pe pagina oficială.

Această bibliotecă ne pune la dispoziție o gamă foarte largă de tipuri de grafuri ce le-am putea utiliza într-o aplicație, de la cel mai simplu graf până la cele mai complexe, oferindu-le, astfel, utilizatorilor un mediu de învățare și punere la încercare a cunoștințelor destul de amplu.

GoJS oferă foarte multe caracteristici avansate, și toate acestea pentru a face interactivitatea cu utilizatorul mult mai amplă. Cele mai importante dintre ele sunt „drag-and-drop” (tragere și plasare), „copy-and-paste” (copiază și lipește), editare de text oriunde pe graf, folosirea unor șabloane pentru crearea anumitor tipuri de grafuri ș.a.m.d.

În continuare vor fi prezentate o mulțime de funcționalități ce vor putea fi folosite dacă o aplicație folosește această bibliotecă. Vom lua ca exemplu un graf creat in urma utilizării **GoJS**, exemplu de graf desenat luat de pe pagina oficială a bibliotecii. Putem spune că aceste funcționalități sunt printre cele mai importante deoarece prin intermediul lor utilizatorul aplicației poate interacționa într-un mod plăcut și simplu cu graful de pe tabla de lucru.

Ce trebuie precizat este că **GoJS** este o bibliotecă ce provine de la **Northwoods Software**, o companie creată de ingineri software special pentru inginerii software ce se ocupă de dezvoltarea aplicațiilor. De aceasta ne putem folosi în totalitate cumpărând-o, astfel având acces la toate funcționalitățile implementate.

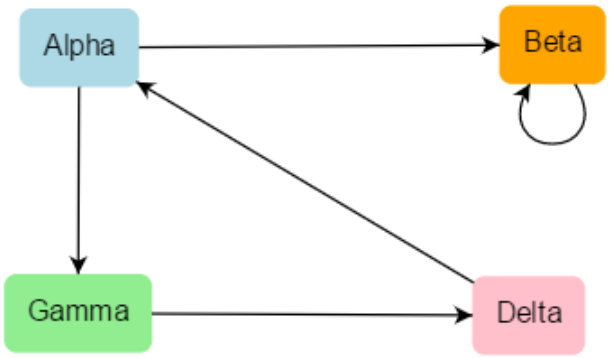

Fig. 2.6: Exemplu minimal de graf desenat cu biblioteca GoJS

Selectarea nodurilor se face prin intermediul cursorului fără alte butoane sau instrumente auxiliare. După selectarea acestora, nodurile sunt scoase în evidență printr-un dreptunghi albastru de jur împrejurul acestora. Muchiile pot fi selectare de asemenea.

În plus, în momentul când se dorește selectarea mai multor elemente din graf nu trebuie decât ținută tasta „Shift” și apoi click pe elementul dorit. Pentru îndeplinirea acestei misiuni este posibilă și o alta metodă, chiar mai familiară pentru utilizatori decât precedenta, și anume, click stânga pe spațiul gol de pe tabla de lucru timp de o secundă și apoi tras de cursor, astfel creând un dreptunghi în interiorul căruia, elementele vor fi selectate. Apăsând tastele „Ctrl+A” selectează toate elementele din graf.

Acțiunea numită „drag-and-drop” ce constă în mutarea unui element dintr-o parte în alta cu ajutorul cursorului este de asemenea prezentă odată cu utilizarea bibliotecii. Cu alte cuvinte aceste controale sunt destul de accesibile utilizatorului, nefiind nevoit să depindă de alte acțiuni adiționale.

Pe lângă gama largă de funcționalități oferită, GoJS se dorește a fi o bibliotecă simplă, ușor de înțeles, ușor de învățat, ușor de utilizat. Pagina oficială oferă atât o secțiune destinată prezentării în detaliu a conceptelor și a API-ului ([11]), cât și o serie de mostre și exemple ([12]) ce pot reprezenta un punct de start pentru noii utilizatori ai bibliotecii.

2.4 Alchemy.js

O altă bibliotecă pentru crearea de grafuri, folosită în aplicații, se numește **Alchemy.js** ([13]). Aceasta este scrisă folosind o altă bibliotecă, numită „**D3**”, ce folosește JavaScript și ne permite sa legăm date sau informații de DOM (din eng. Document Object Model). Spre exemplu, se poate folosi **D3** pentru a genera în HTML un tabel doar dintr-un vector de numere.

Alchemy.js a fost construit în special pentru inginerii IT în scopul de ai ajuta în crearea de aplicații ce au nevoie de vizualizarea unor grafuri. Nu este necesar o cantitate mare de cod pentru

crearea unor grafuri în **Alchemy.js**. Decorarea aplicației se face printr-o simplă suprascriere a configurărilor setate în mod implicit.

Deoarece **Alchemy.js** a fost construit folosind **D3**, aplicația principală poate fi extinsă cu ușurință cu oricare altă proprietate aparținând bibliotecii de bază. Aici, un graf este salvat sub forma unui fișier JSON.

Sunt foarte multe acțiuni predefinite în această bibliotecă atât pentru noduri, cât și pentru muchii, acestea putând fi atașate de un anumit eveniment din interfața utilizatorului, precum un buton. În continuare este exemplificat rezultatul creării unui graf cu ajutorul **Alchemy.js**

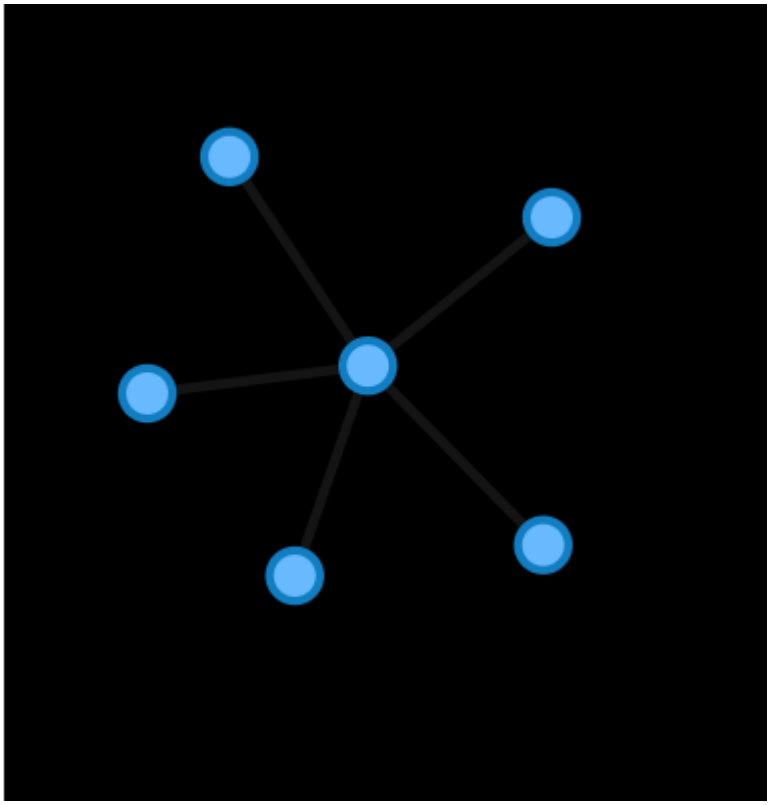

Fig. 2.7: Exemplu de graf desenat cu biblioteca Alchemy.js

De asemenea se poate regăsi, urmărind următorul link, o întreagă aplicație creată cu biblioteca **Alchemy.js: [14].**

2.5 Cytoscape.js

Cytoscape.js ([15]) este o altă bibliotecă JavaScript ce oferă suport pentru lucrul cu grafuri într-un mod interactiv. Este un proiect open-source ce a fost început la Centrul Donnelly al Universității din Toronto. Pe lângă funcționalitățile de desenare și vizualizare, include și implementări ale celor mai des întâlniți algoritmi din teoria grafurilor.

O altă caracteristică a acestei biblioteci este compatibilitatea nu doar cu browser-ele desktop cât și cu cele mobile. Oferă posibilitatea de a genera evenimente bazate pe interacțiunea utilizatorului cu editorul de grafuri, ce pot fi la rândul lor prelucrate în aplicații mobile. Conform paginii oficiale, **Cytoscape.js** este compatibilă cu: broswerele desktop, CommonJS/Node.js, AMD/Require.js, jQuery, npm, Bower, Meteor/Atmosphere, The R language [16] via RCyjs [17].

Experiența utilizatorului este un aspect important atunci când vorbim despre aplicații interactive. La acest capitol, **Cytoscape.js** prezintă suport pentru o gamă variată de gesturi precum: drag-and-drop (touch și desktop), utilizarea degetelor pentru zoom in, zoom out, selectarea și deselectarea elementelor, utilizarea tastelor de control pentru selecții multiple etc.

Pentru stilizarea grafurilor, biblioteca încearcă să urmeze pe cât de mult posibil convențiile de nume din CSS, însă oferă și câteva proprietăți specifice. De asemenea există suport și pentru crearea de animații ce pot fi de folos în expunerea algoritmilor din teoria grafurilor. Elementele de animație sunt diverse, pornind de la efectul în sine până la proprietăți de durată, întârziere. Mai multe detalii sunt oferite în documentația oficială.

Referitor la performanța în procesul de randare, autorii bibliotecii atrag atenția asupra riscului ca aplicația să devină foarte înceată odată cu creșterea numărului de elemente ale grafului și complicării elementelor stilistice. Este un aspect ce trebuie luat în considerare deoarece poate degrada gradul de satisfacție al utilizatorilor.

2.6 Concluzii

Aplicațiile prezentate în primă fază în acest capitol pot fi extrem de folositoare pentru orice persoană care dorește sa dobândească cunoștințe elementare din teoria grafurilor. Am putea spune că acestea reprezintă o rampă de lansare spre cunoașterea unor aspecte mult mai complexe, dar, în același timp, și mai intrigante și interesante. Cât despre celelalte tehnologii prezentate ulterior și dedicate desenării grafurilor, toate au avantajele și dezavantajele lor în a le folosi. În general aceste biblioteci necesită achiziționare astfel încât inginerii software sa poată beneficia de ele în totalitate. Ceea ce mi-am propus este ca să fac acea aplicație care nu are rolul doar de inițiere în teoria grafurilor, ci poate ajuta la înțelegerea și a altor algoritmi mai complecși, și totodată este accesibil tuturor fiind utilizabil într-un mod satisfăcător (user-friendly).

Capitolul 3

Tehnologii folosite

3.1 Angular 2

Angular 2 este un framework open source dedicat pentru construirea de aplicații in HTML și JavaScript. Spre deosebire de Angular 1.x, Angular 2 și Typescript ne aduc o dezvoltare a aplicațiilor web într-un mod orientat obiect. Potrivit inginerului director de la Google, 1.3 milioane de developeri folosesc AngularJS și 300.000 folosesc deja Angular 2. La sfârșitul lui 2014 Google a anunțat ca Angular 2 va fi o schimbare completa pentru AngularJS, astfel ei au creat un nou limbaj, „AtScript", ce era folosit la scrierea de aplicații in Angular 2. Dar, ulterior, Microsoft a fost de acord sa adauge suport pentru adnotări in limbajul TypeScript, astfel apărând chiar Angular 2.

În plus, cei ce au creat Angular au integrat încă un alt produs Microsoft, și anume biblioteca RxJS. Angular 2 nu e un framework bazat pe MVC (Mode View Controller), ci unul bazat pe componente. O aplicație web scrisă in acest framework este formata dintr-un arbore de componente. Randarea automata se face prind legarea (bind) template-ului de vectorul de componente primit de la server (sursa: [18]).

Aplicația creată, scrisă în Angular 2, folosește Typescript, lucru recomandat și de acest framework. Typescript este o extensie a limbajului JavaScript care permite adnotarea tipurilor de date. Programatorii familiari cu limbajele de programare orientate pe obiect vor găsi Typescript-ul ușor de utilizat. Astfel, scopul acestuia este de a crea aplicații web cu mult mai repede. În schimb, acesta poate fi văzut ca un impediment în procesul dezvoltării de aplicații daca vrei ca totul sa fie scris în Typescript. De asemenea se poate scrie în Typescript doar anumite părți din aplicație, unde acesta chiar poate fi util și își poate lăsa amprenta în eficiența procesului dezvoltării aplicației.

O funcționalitate importantă, pe care am și folosit-o în aplicația mea, specifică Angular 2 este cea care ne permite crearea aplicațiilor web pe o singură pagină (Single Page Application). Cu ajutorul acesteia, aplicația nu se va încărca decât o singură dată permițându-ne să navigăm printre funcționalități fără reîncărcarea paginii.

Este mai indicat să folosim framework-ul Angular 2 decât vreo versiune anterioară acestuia deoarece, în primul rând, este mai rapid, structura codului este simplificată, are mai puține concepte, și astfel este mai ușor de înțeles. Ca și dezavantaj în folosirea acestei tehnologii poate fi faptul că aplicația poate deveni înceată relativ atunci când vrem să afișăm date foarte multe. Angular 2 folosește, de altfel, NPM, cunoscut ca și Node Package Manager, pentru a lucra cu repertoriul de biblioteci open-source. NPM poate fi folosit pentru a descărca aceste dependențe și pentru a le atașa proiectului nostru (tutorial angular2: [19]).

3.2 Vis.js

Vis.js este o bibliotecă JavaScript open-source dinamică de desenare și vizualizare a graficelor. Biblioteca aceasta a fost creată pentru utilizatori sau developeri cu scopul de a fi ușor de utilizat. Ce se poate desena cu această bibliotecă sunt:

- Grafuri/Networks (afișare și desenare de grafuri formate din noduri și muchii);
- DataSet și DataView (un ansamblu de date bazat pe cheie/valoare);
- DataView (un „view” filtrat sau formatat al DataSet-ului);
- Graph2d (reprezentarea prin linii sau bare a unor date – grafic – Fig. 3.1);
- Graph3d (reprezentare de date într-un grafic 3d – Fig. 3.2);
- Timeline (reprezentarea a diferite tipuri de date într-o cronologie Fig. 3.3). [20]

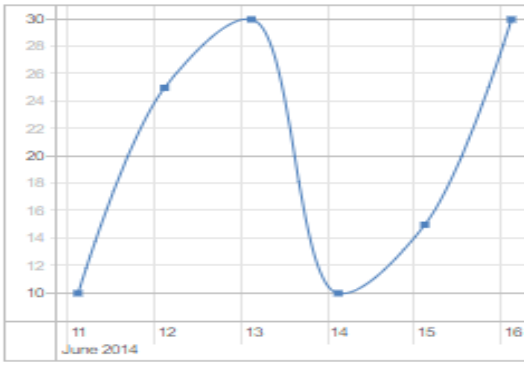

Fig. 3.1: Exemplu de grafic 2D desenat cu Vis,js

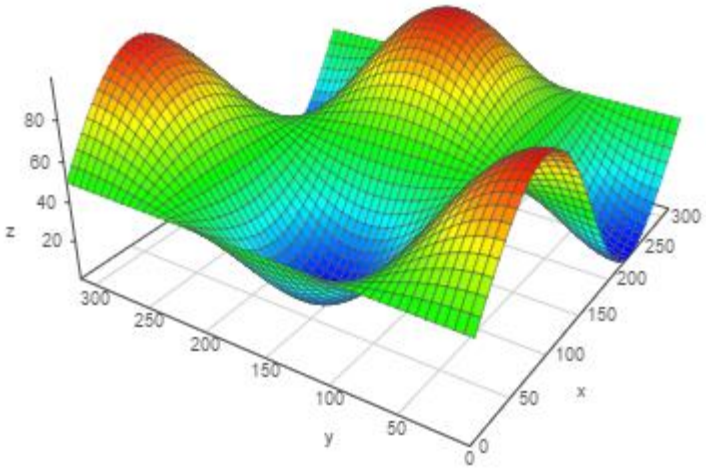

Fig. 3.2: Exemplu de grafic 3D desenat cu Vis.js

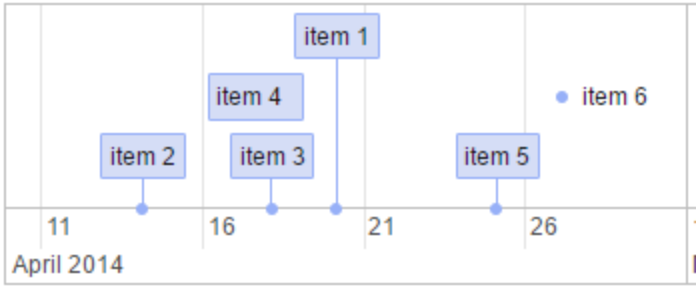

Fig. 3.3: Exemplu de cronologie desenată cu Vis.js

Mai sus sunt ilustrate 3 tipuri de desenări pe care le suportă această bibliotecă. Cel de-al patru-lea tip, și cel utilizat în această aplicație, este desenarea de grafuri, alcătuite din noduri și muchii. Am ales această bibliotecă să o folosesc pentru că este ușor de utilizat atât de developeri, creând aplicația, cât și de utilizatori accesând-o. Rezultatul după desenare este unul demn de aplicații mari, utilizabile la un rang înalt. În principal, acest lucru este datorat fiabilității grafurilor create și ușurinței modului de a interacționa cu acestea. Poate suporta diferite forme, stiluri, culori, mărimi iar numărul foarte mare de noduri nu este o problemă pentru **Vis.js**. Aceasta poate lucra cu noduri și muchii de până la o mie ca număr, iar pentru a lucra cu mult mai multe, rețeaua sau graful poate suporta clusterizarea. Pentru randarea grafului, **Vis.js** folosește *HTML canvas*.

Inițial, biblioteca **Vis.js** a fost conceputa de o firma olandeză cu numele de *Almende B.V.* [21]. Aceasta din urmă își are sediul în Rotterdam și se ocupă, în special, cu dezvoltarea de produse software în domeniul ICT (*Information and communication*). Fiind o tehnologie open-source, oricine pasionat de programare și dorește îmbunătățirea acestei biblioteci poate contribui prin implementarea unor funcționalități noi sau îmbunătățirea celor prezente ([22], [23]).

Odată cu versiunea 4.0, s-au adăugat module individuale și specifice componentelor grafurilor. Acestea au la rândul lor, pe pagina oficiala **Vis.js**, propriile documentații în care sunt descrise opțiunile, metodele și evenimentele de care dispun. În continuare vom elabora strict modulele, metodele și evenimentele utilizate în aplicația noastră.

De obicei, în momentul în care dorim sa creăm un graf cu ajutorul acestei biblioteci, vom crea o funcție „*draw()*" în care vom inițializa parametrii de care are nevoie funcția pusă la dispoziție de **Vis.js** ca mai apoi sa o apelăm. Parametrii nu puteau fi mai clari sau mai evidenți de atât, astfel că avem nevoie de un container (element în DOM unde va fi plasat graful desenat), datele grafului (acest parametru este alcătuit din doi vectori importanți în care vom stoca nodurile și muchiile) și, nu în ultimul rând, opțiunile grafului cu ajutorul cărora vom personaliza graful din momentul creării acestuia. Despre aceasta din urmă vom vorbi ulterior mai în amănunt. Imediat după inițializarea acestor date esențiale putem apela în sfârșit metoda de creare și inițializare a grafului (de exemplu: „`myGraph = new vis.Network(container, data, options);`").

3.2.1 Module

Acum că știm cât de ușor și practic este să creăm un graf, putem vorbi despre numeroasele opțiuni pe care i le putem atașa grafului astfel încât să-l modelam exact după dorințele noastre. Opțiunile se stochează într-un obiect cu ale sale proprietăți care reprezintă în biblioteca **Vis.js** *modulele*:

- **Configurație**

Acest modul poate fi integrat în graful nostru prin declararea proprietății „*configure*" în opțiuni. Aici ne putem genera un tabel cu diferite controale care modifica aspectul grafului în mod general pentru toate nodurile sau toate muchiile. Acest tabel se poate personaliza în funcție de ce își dorește developer-ul sa aibă în aplicația sa (Fig. 3.4).

Fig. 3.4: Exemplu de configurație atașata grafului

Acest modul vine și el la rândul său cu o altă proprietate similară cu parametrul funcției de creare a grafului, și anume „*container*”. Această proprietate este mandatorie și trebuie definită pentru a spune opțiunilor locul în care să plaseze tabelul de configurări. Ca și tip de date primește numai elemente existente din DOM.

- **Muchii**

Modulul muchii îl integrăm în opțiuni cu proprietatea numită „*edges*”. Aceasta este de asemenea un obiect ce conține diferite opțiuni pentru muchii. Acestea se aplică în mod global pentru toate muchiile în momentul creării grafului. Ulterior, opțiunile pot fi suprascrise pentru fiecare în parte adăugând aceeași proprietate în vectorul de „*edges*” din „*data*”, de la inițializarea grafului, la obiectul muchiei respective (spre exemplu: „`data.edges.labelHighlightBold = true;`”).

În **Vis.js** muchiile sunt stocate, după cum am menționat și mai devreme, într-un vector conținut de un obiect „*data*”. Vectorul conține obiecte muchii. Aceste obiecte, pentru a îndeplini condiția minimă de a fi muchii, trebuie sa aibă anumite proprietăți:

- *„id”* – un număr de identitate unic cu ajutorul căruia identificăm muchia;
- *„from”* – id-ul unic al nodului dinspre care pornește muchia;
- *„to”* – id-ul unic al nodului înspre care indica muchia.

Pe lângă aceste proprietăți minime se mai poate adaugă oricare alta din modulul „*edges*” din „*options*”. Important: id-ul este specific doar fiecărei muchii în parte.

- **Noduri**

Nodurile reprezintă un alt modul important în aplicația noastră ce este integrat în „*options*” ca o proprietate sub numele de „*nodes*”. De asemenea, toate caracteristicile pentru noduri declarate aici vor fi aplicate la nivel global pentru toate vârfurile grafului la momentul creării acestuia. Aici se aplică aceeași regulă ca și la muchii, și anume că proprietățile nodurilor declarate la început prin „*options*” pot fi suprascrise mai târziu adăugând aceeași proprietate cu o valoare diferită, în vectorul „nodes” folosit pentru stocarea datelor nodurilor (spre exemplu: „`data.nodes.borderWidth = 3`”).

Spre deosebire de muchii, obiectele din vectorul „nodes” pot îndeplini condiția de a fi noduri doar prin a furniza un id unic pentru fiecare dintre ele. Daca proprietatea „*label*” nu este definită, atunci, pe nod va apărea id-ul său, neîncurcând cu nimic decursul aplicației.

- **Manipulare**

Funcționalitățile principale folosite din acest modul, ce poate fi definit în „*options*” ca „*manipulation*”, sunt:

- *„addNode”* **–** adăugarea unui nod;
- *„addEdge”* **–** adăugarea unei muchii;
- *„editNode”* **–** editarea unui nod;
- *„editEdge”* **–** editarea unei muchii;
- *„deleteNode”* **–** ștergerea unui nod;
- *„deleteEdge”* **–** ștergerea unei muchii;

Având aceste proprietăți setate pe „*true*” în obiectul „*options*”, toate acțiunile principale, și de altfel esențiale, pentru crearea unei aplicații ce lucrează cu grafuri vor fi posibile de executat, permițându-ne sa manipulam graful setat inițial cu anumite date (un vector inițial de noduri și unul de muchii).

Aceste proprietăți de foarte mare ajutor aplicației mai ai posibilitatea de a fi definite chiar și ca funcții și nu numai ca valori booleene. Să luam ca exemplu următoarele linii de cod:

```
var options = {
      manipulation: {
            addNode: function(nodeData, callback) {
                  nodeData.label = `hello world`;
                  callback(nodeData);
            }
      }
}
```

(sursa: [24])

După cum se poate observa, în momentul în care utilizatorul efectuează acțiunea de adăugare a unui nod (în acest exemplu), putem rula o funcție în care putem afișa un mesaj sau face altceva. Acest lucru este extrem de folositor în procesul de dezvoltare a aplicației. Evident, putem aplica aceeași metodă de atașare a unei funcții pentru fiecare dintre acțiunile menționate mai sus.

- **Amplasare**

Ce merită menționat aici este că un developer care folosește **Vis.js** își va putea poziționa graful într-un mod definit de el în proprietățile modulului „*layout*" din „*options*":

- „*improvedLayout*" **–** această proprietate setată pe „*true*" face ca graful nostru să se poziționeze conform unui algoritm predefinit din bibliotecă;
- „*hierarchical*" **–** probabil cea mai importantă proprietate pentru aplicația noastră din acest modul; dacă este setată pe „true" graful nostru se va poziționa în așa fel încât să arate ca un arbore;
- „*randomSeed*" **–** se aplică doar atunci când nu folosim vizualizarea ierarhică, și face ca să ne poziționeze graful într-un mod aleatoriu;

Proprietatea „*hierarchical*" reprezintă, de fapt, un obiect cu alte proprietăți cu ajutorul cărora putem personaliza afișarea ierarhică a grafului nostru după bunul plac. Acest modul poate face diferența într-o aplicație deoarece felul cum sunt plasate nodurile și muchiile într-un graf, vorbind aici despre arbori în special, are o mare importanță în vizualizarea acestuia.

- **Fizică**

În primul rând, acest modul va fi amplasat în opțiuni ca un obiect și îl vom denumi „*physics*". Aici vorbim mai mult despre consecințele interacțiunii utilizatorului cu graful în sine. Cei care au implementat biblioteca **Vis.js** s-au gândit că ar fi o idee bună să introducă o opțiune de acest gen care daca la acordul utilizatorului, graful nostru sa se comporte după anumite legi ale fizicii, lăsând astfel o senzație plăcută și primind numai retroacțiuni („feedback-uri") pozitive în urma utilizării aplicației.

De asemenea, și modulele „*edges*" și „*nodes*" au la rândul lor, fiecare, proprietăți ce poartă numele „*physics*", pe când acest modul „physics" din opțiuni se adresează grafului în întregime.

Acestea au fost cele mai importante module de opțiuni pe care le-am folosit în aplicație. Mai există alte doua module și anume „*groups*” și „*interaction*” ce se pot atașa ca opțiuni pentru graf. După cum am văzut, sunt bine puse la punct și sunt de mare ajutor celor care vor sa folosească biblioteca **Vis.js.** Un lucru ce trebuie menționat este că Vis.js nu duce lipsă de documentație tehnică amănunțită despre aceste module, în care sunt explicați pașii de urmat pentru atribuirea unei opțiuni în graf și rezultatele la care sa ne așteptăm ([25]).

Din moment ce acum știm cum se creează un graf setând opțiunile necesare în imaginea următoare avem un exemplu minimal creat în biblioteca **Vis.js**:

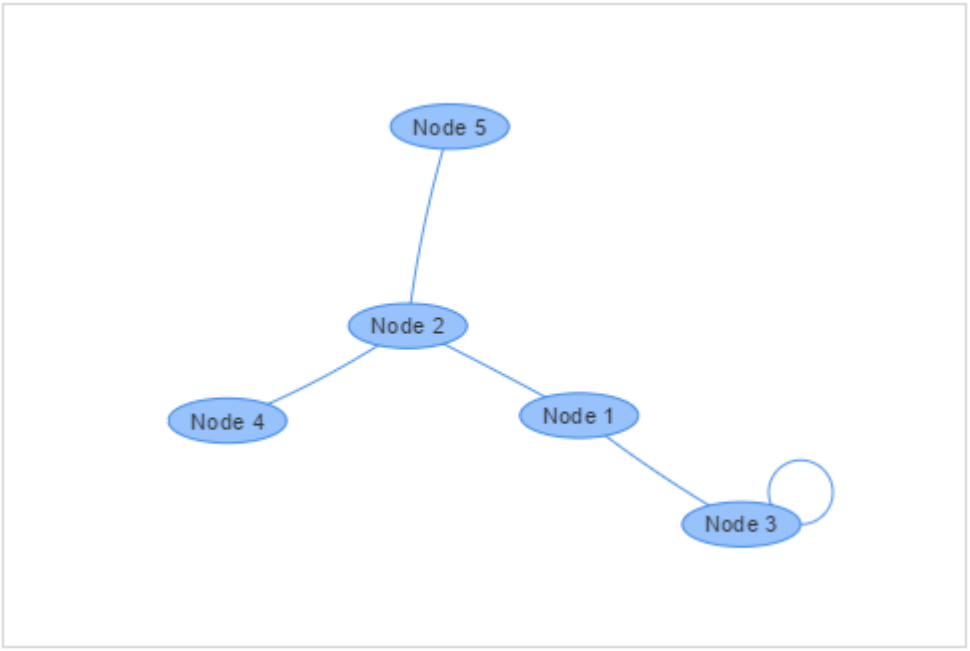

Fig. 3.5: Exemplu de graf desenat cu Vis.js

3.2.2 Metode

În continuare vom prezenta succint câteva metode din API pe care, de altfel, le-am și folosit în aplicația practică. Ele sunt descrise tehnic amănunțit și pe pagina oficială **Vis.js [26]**, iar pe lângă asta, voi da și câteva exemple de situații în care ar putea fi folosite.

Din cauză că pe pagina web, metodele sunt prezentate pe categorii (ex: globale) și aplicația noastră nu folosește prea multe dintre ele, vom spune la fiecare dintre ele din ce categorie fac parte. Așadar, principalele metode sunt:

- *„destroy()”* – folosit pentru distrugerea completă a grafului desenat până în momentul respectiv. Folositor pentru utilizatori atunci când doresc ștergerea în totalitate a tablei de lucru. Face parte din categoria metodelor globale.

- *„setData(data)”* – noul graf va fi reprezentat de datele primite prin parametrul *„data”*, acesta reprezentând un obiect ce conține un vector de noduri și unu de muchii. Dacă există un graf înainte de a se apela această funcție, datele se suprascriu dar opțiunile rămân aceleași. Folositor când utilizatorul dorește importarea datelor unui graf. Face parte din categoria metodelor globale.

- *„setOptions(options)”* – se setează opțiunile grafului utilizând parametrul *„options”*. Prin apelarea acestei metode opțiunile prezente până în momentul respectiv nu vor fi șterse, ci vor fi actualizate. Este o metoda destul de practică pentru developeri deoarece nu sunt nevoiți să salveze opțiunile precedente într-o altă variabilă auxiliară. Face parte din categoria metodelor globale.

- *„on(event, function)”* – atașează o funcție de un eveniment ce poate avea loc în aplicație. Despre evenimente vom vorbi ulterior. Folosită ca un ascultător de eveniment util developerilor. Face parte din categoria metodelor globale.

- *„off(event, function)”* – îndepărtează ascultătorul de eveniment dacă acesta a fost setat înainte. În principiu este exact opusul metodei *„on”* și este, de asemenea, o metodă globală.

- *„redraw()”* – metodă folosită pentru a redesena graful și este din categoria de desenare.

- *„enableEditMode()”*, *„disableEditMode()”*, *„addNodeMode()”*, *„editNode()”*, *„addEdgeMode()”*, *„editEdgeMode()”*, *„deleteSelected()”* – toate acestea sunt funcții de manipulare a grafului fără a se folosi de interfața grafică.

Acestea au fost metodele puse la dispoziție de API de care m-am decis să mă folosesc în aplicația practică. Lista completă și detaliată poate fi găsită pe pagina **Vis.js [26]**.

3.2.3 Evenimente

Vis.js vine cu o serie de evenimente disponibile în API de care ne putem folosi în diferite situații. Acestea sunt declanșate de modulul amintit mai devreme, și anume, „*interaction*” și sunt în strânsă legătura cu acțiunile utilizatorului. După cum am amintit și în secțiunea metodelor, funcția „*on(event, function)*” se folosește de aceste evenimente puse la dispoziție, oferind utilizatorilor acestei biblioteci o ușurință remarcabilă în dezvoltarea aplicațiilor.

Pe cele mai importante le vom descrie succint în rândurile ce urmează:

- *„click”* – unul dintre cele mai comune evenimente ce pot avea loc în aplicații web. Este declanșat atunci când utilizatorul apasă click stânga oriunde pe tabla de lucru. De asemenea acest eveniment furnizează un obiect cu anumite proprietăți legate de ce a selectat utilizatorul:

```
{
      nodes: [Array of selected nodeIds],
      edges: [Array of selected edgesIds],
      event: [Object] original click event,
      pointer: {
            DOM: {x:pointer_x, y:pointer_y},
            canvas: {x:canvas_x, y:canvas_y}
      }
}
```

(sursa: [27])

- *„doubleClick”* – atunci când este declanșat acest eveniment, de fapt se declanșează trei evenimente în ordinea următoare: *„click”*, *„click”*, *„doubleClick”*. Pentru evitarea acestui lucru trebuie verificat timpul dintre cele doua click-uri.

- *„select”* – spre deosebire de *„click”*, acest eveniment se declanșează doar în momentul în care ori un nod, ori o muchie au fost selectate. Legate de *„select”*, exista și evenimente separate pentru noduri și muchii: *„selectNode”*, *„selectEdge”*, *„deselectNode”*, *„deselectEdge”*.

- *„dragging”* – eveniment declanșat în momentul în care folosim tehnica „drag-and-drop” asupra nodurilor. Putem utiliza, de asemenea, *„dragStart”* ce se declanșează la început, sau *„dragEnd”* la sfârșit.

Lista completă cu toate evenimentele disponibile odată cu folosirea bibliotecii Vis.js poate fi găsită pe pagina oficială [27].

3.3 ng-sidebar

„**ng-sidebar**” este o altă tehnologie folosită în aplicație și reprezintă o componentă ce se poate folosi în Angular 2 și oricare versiune mai nouă a acestui framework. După ce ne setăm aplicația să folosească biblioteca „**ng-sidebar**” și ne alegem versiunea acesteia, putem să o folosim astfel:

Într-un document HTML, pe pagina unde vrem sa adăugam bara laterală (din eng.: „*sidebar*”) vom adaugă următoarele linii de cod:

pagină.html:

```
<!−−Container pentru bara/bare laterală/laterale−−>
<ng-sidebar-container>
      <ng-sidebar [(opened)] = „_deschis”>
            <p>Conținut bară laterală</p>
      </ng-sidebar>

      <!−Conținut pagină−−>
      <button (click) = „_afișBaraLaterală()”>Afișează bara laterală</button>
</ng-sidebar-container>
```

Component.ts:

```
class Component {
      private _deschis: boolean = false;

      // funcție de afișare bară laterală
      private _afișBaraLaterală() {
            this._opened = !this._opened;
      }
}
```

(sursa: [28])

Deschiderea și închiderea barei laterale se face într-un mod plăcut fiind acompaniată de o animație. Componenta „**ng-sidebar**” vine cu o serie de atribute predefinite de care ne putem folosi, cele mai utilizate în aplicația noastră fiind:

- *„opened”* – primește valori booleene și decide dacă bara noastră laterală este deschisă sau închisă. Putem manipula valoarea acestei variabile, spre exemplu, prin atașarea unei funcții de un buton ce o schimbă prin simpla apăsare a acestuia.

- *„mode”* – poate lua următoarele valori:

    - *„over”* – bara laterală se va deschide peste celelalte elemente;

    - *„push”* – bara laterală va împinge tot conținutul paginii în partea opusă deschiderii făcând ca toate elementele să se restrângă;

    - *„slide”* - bara laterală va împinge tot conținutul paginii în partea opusă deschiderii făcând ca toate elementele ce nu încap în pagină să iasă în afara acesteia;

- *„position”* – atribut ce determină poziția de unde să apară bara laterală. Poate lua următoarele valori: *„left”*, *„right”*, *„top”*, *„bottom”*, *„start”*, *„end”*.

Acum că știm cum să folosim bara laterală în aplicația noastră Angular 2, să vedem cum arată pe pagina unde am atașat-o:

Fig. 3.6: Exemplu minimal de bară laterală „ng-sidebar”

O pagină web utilă unde se pot testa toate funcționalitățile acestei componente este [29]. Link-uri utile găsim tot acolo precum:

- pagina pachetului *„npm”*, **„ng-sidebar”**;
- codul sursă al componentei, disponibil pe *„GitHub”*;
- codul sursă al exemplului, disponibil pe *„GitHub”*.

3.4 Concluzii

Aruncând o privire la prima tehnologie, Angular 2, putem spune că este una dintre cele mai în vogă la ora actuală, iar pentru o aplicație bazată numai pe front-end este o alegere foarte bună. Există numeroase oportunități ce ne oferă acest framework, una dintre acestea fiind faptul că putem folosi și alte biblioteci construite pe baza acestei tehnologii, Angular 2, astfel încât, acestea să funcționeze armonios împreună. Luând exemplul de față, putem spune acest lucru chiar despre Angular 2 și „ng-sidebar”, cea din urma fiind o componenta creată pe baza principiilor celei dintâi. Și nu în cele din urmă, trăgând linie și adunând avantajele cu dezavantajele fiecărei biblioteci de desenare grafuri putem observa motivul alegeri lui „Vis.js” în ciuda celorlalte prezentate în capitolul precedent. Nu putem spune că este cea mai bună, dar este cu siguranță printre ele.

Capitolul 4

Descrierea aplicației LearnGraphs

4.1 Analiză și proiectare

Această aplicație, „**LearnGraphs**”, a fost implementată cu gândul de a crea o unealtă pentru profesorii din liceu, scopul acesteia fiind să inițieze elevii într-un capitol mare și important, „Grafuri și Arbori”. Pentru cei care doresc să continue în informatică după liceu, felul în care ei înțeleg acest capitol determină capacitatea lor de comprehensiune mai târziu pentru subiectele cu mult mai complexe pe care le vor întâlni pe băncile facultății.

După cum spuneam, „**LearnGraphs**” este o unealtă perfectă pentru profesorii aflați în procesul de predare a materiei, aceștia putând exemplifica algoritmii pentru grafuri sau arbori într-un mod vizual și plăcut, acesta dând de cele mai multe ori roade când vine vorba despre acumularea informațiilor. Însă, în aceasta ecuație mai intervine un factor, și anume timpul prea scurt de predare. Cadrul didactic are, de obicei, un număr de zile disponibil sa-l consume pentru a trece printr-o anumită materie. De aceea mulțimea persoanelor care pot folosi „**LearnGraphs**” se poate lărgi, fiecare elev având acces, de acasă, la aceasta aplicație. Astfel, copiii din liceu vor putea sa învețe și să aprofundeze și de acasă în caz că timpul petrecut la scoală nu le este de ajuns.

„**LearnGraphs**”, fiind o aplicație ce folosește Angular 2, de asemenea are un fișier unde declarăm pachetele folosite. Astfel ele sunt descărcate direct în aplicația noastră folosind comanda „*npm install*”. Deci, aplicația se instalează foarte ușor iar pentru a porni-o nu trebuie rulată decât următoarea comandă: „*npm start*”. În acest moment ne apare un mesaj în consola care ne transmite către ce port putem accesa aplicația. Astfel, acum nu ne mai rămâne de făcut decât să deschidem un navigator de internet iar în bara de adresa sa introducem „*http://localhost:port*”

Pentru exportarea și importarea unui graf de către utilizatorul aplicației am folosit un format ușor de interschimbare a datelor: JSON (JavaScript Object Notation – Notația Obiect JavaScript). Ochiul uman citește cu ușurință acest format și, astfel, îl face și ușor de scris de către om, acest lucru fiind motivul principal în alegerea lui pentru funcționalitatea de „*export / import*”. Pe lângă toate acestea, și mașinile analizează și generează acest format cu ușurința. Toate acestea fiind spuse, JSON este formatul perfect pentru o astfel de funcționalitate în aplicația noastră. Despre „export / import” vom vorbi ulterior în acest capitol. Exemplu format JSON:

```
var animal = {
      „nume”: „Rex”,
      „tip”: „câine”,
      „sex”: „m”,
      „vârsta”: „1”
};
```

În continuare, avem ilustrată o diagramă „*Use Case*” a aplicației noastre ce ilustrează utilizarea acesteia fie de profesor, fie de elev, și diferite acțiuni pe care le pot face:

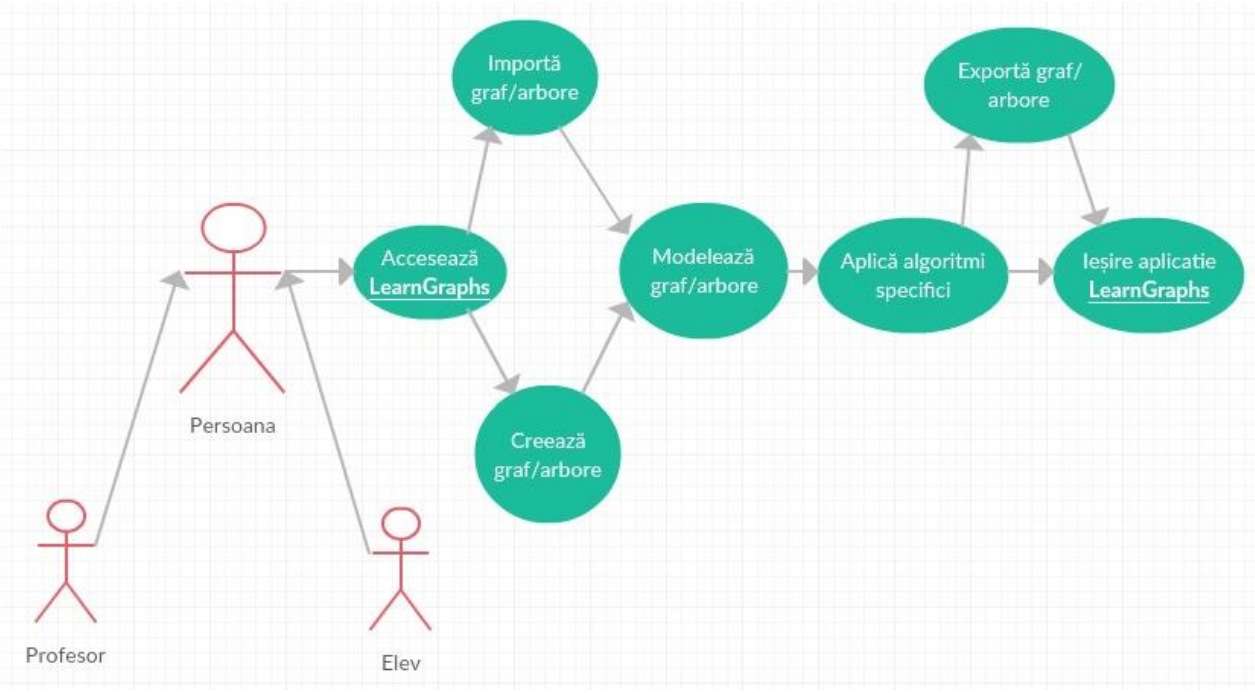

Fig. 4.1: Diagramă „*Use Case*” a aplicației **LearnGraphs**

(creată pe [30])

Pentru o înțelegere mai bună a aplicației **LearnGraphs**, despre cum interacționează componentele principale ale acesteia, am creat o diagramă a arhitecturii. Principalele elemente reprezentate sunt atât componentele din Angular 2 create prin intermediul limbajului TypeScript, cât și tehnologiile utilizate de acestea:

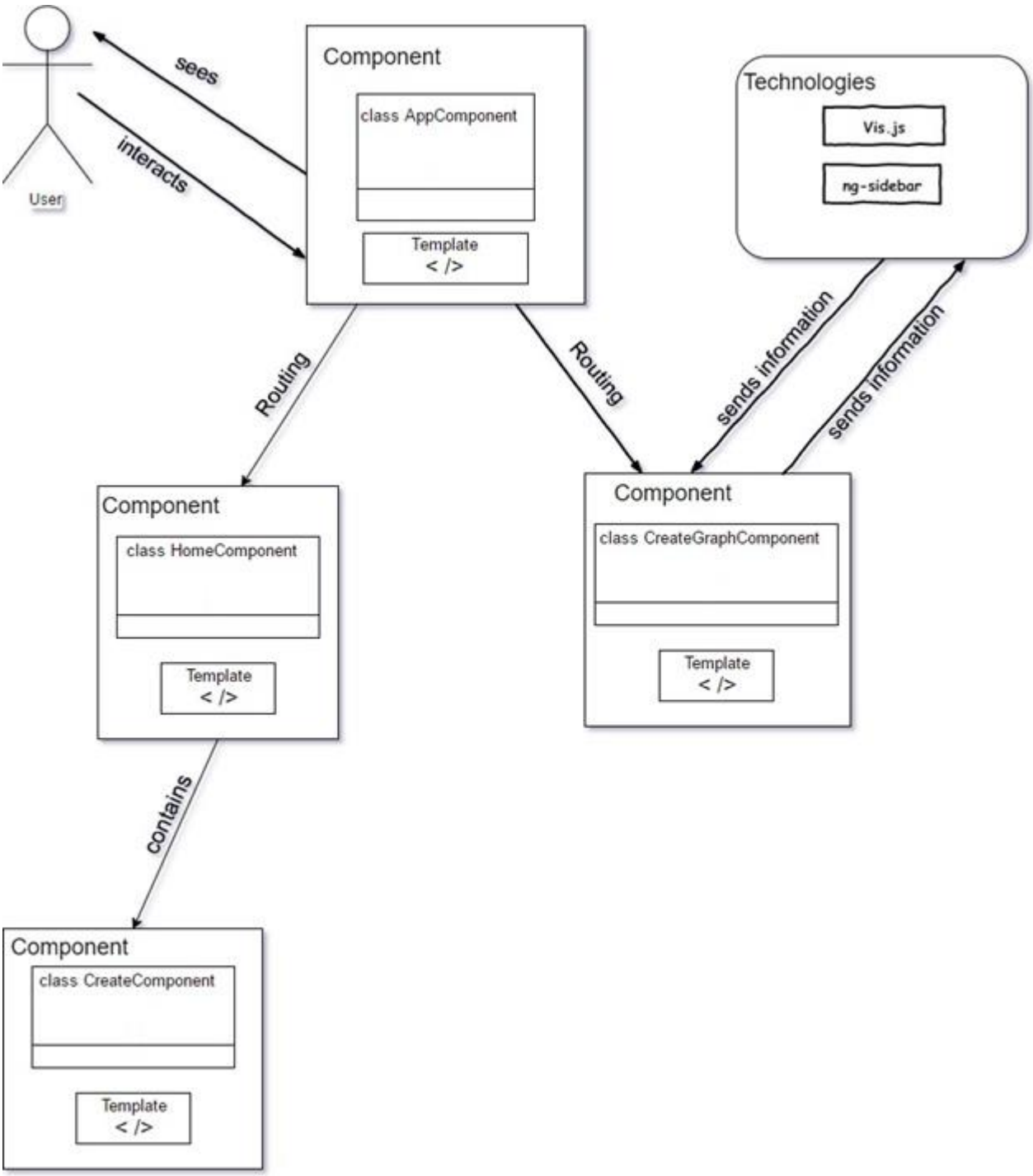

Fig. 4.2: Arhitectura aplicației LearnGraphs

(creată pe [31])

4.2 Implementare

4.2.1 Routing

LearnGraphs a fost concepută cu ideea de a fi o aplicație ce folosește numai tehnologii de front-end. Una dintre acestea fiind Angular 2. Acest framework este unul dintre cele mai folosite la ora actuală pentru că ne oferă multe oportunități. Una dintre acestea este posibilitatea de a-ti crea aplicația proprie doar pe o singură pagină, încărcarea acesteia având loc o singură dată. Aici, acest fenomen poartă denumirea de „*routing*”.

În aplicația noastră vom lua exemplul următor: după cum se poate observa și în arhitectura aplicației din capitolul precedent, „*routing*”-ul se face între două componente, „*HomeComponent*” și „*CreateGraphComponent*”. Acum vom avea nevoie de o directivă pusă la dispoziție de Angular 2, și anume „*router-outlet*”. Prin intermediul acesteia, putem atașa elementul „<*router-outlet*>” aplicației noastre, locul în care va fi amplasat fiind, în același timp, locul în care poate apărea fie „*HomeComponent*”, fie „*CreateGraphComponent*”, astfel, fără ca pagina să se reîncarce.

app.component.ts

```
// componenta aplicației
@Component({
      ...
      template: `<router-outlet></router-outlet>`
      ...
})
```

routes.ts

```
...
// fișier ce conține rutele spre componente
const appRoutes:Routes = [
      {path: 'create', component: CreateGraphComponent},
      {path: '', component: HomeComponent, pathMatch: 'full'}
];
export const routing: ModulesWithProviders = RouterModule.forRoot(appRoutes);
```

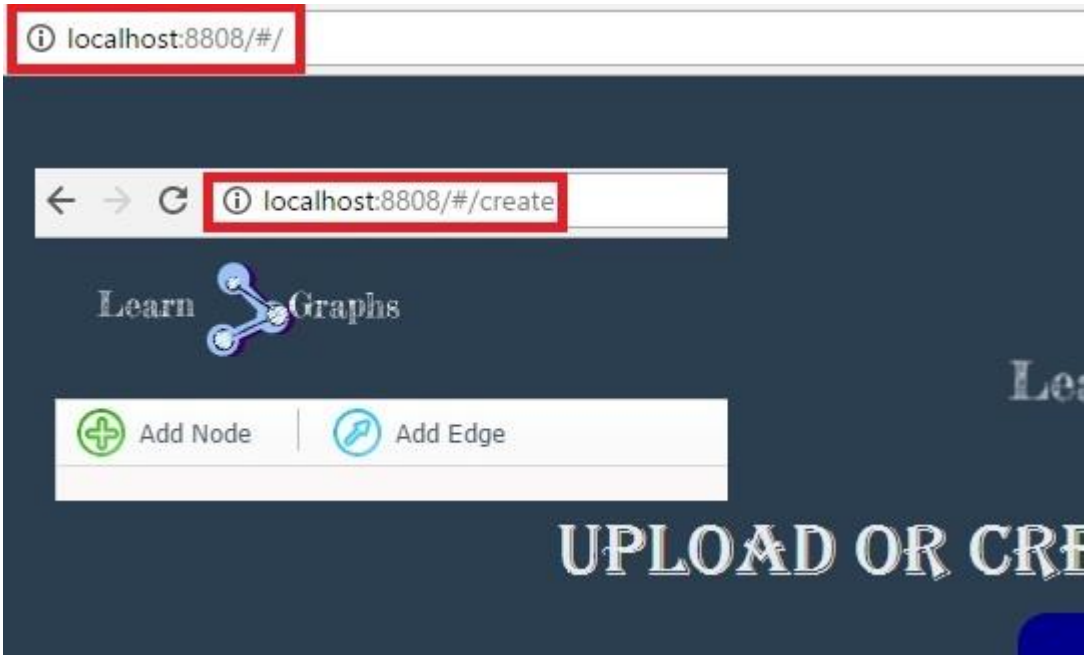

Fig. 4.3: „*Routing*” în LearnGraphs

Cu scopul de a evita erori de tipul „*404 not found*”, am folosit „*HashLocationStrategy*” în așa fel încât router-ul să folosească „*/#/*”, de unde și apariția acestuia în link.

Pentru o tranziție plăcută, din punct de vedere vizual, de la o componentă la alta, fiecăreia dintre ele i-am adăugat câte o animație ce face o componenta sa dispară pe partea de sus a paginii în timp ce cealaltă apare tot pe acolo:

home.component.ts

```
...
// componenta HomeComponent
@Component({
      ...
      animations: [routerTransitions()]
      ...
})
```

4.2.2 Editarea grafului

Funcționalității de editare a nodurilor sau muchiilor i-am mai adăugat un meniu din care utilizatorul poate alege diferite opțiuni pentru graf. Acest meniu este conținut de un element de tip bară laterală, creat cu ajutorul tehnologiei „*ng-sidebar*”.

Editarea opțiunilor nodurilor sau ale muchiilor se face cu ajutorul tipului de dată disponibil odată cu utilizarea bibliotecii **Vis.js**, și anume „*Dataset*”. Vectorul de muchii și, respectiv, cel de noduri, sunt fiecare salvați în câte o variabila de acest tip. Astfel, cu ajutorul constructorului ce primește ca parametru un vector fie de muchii, fie de noduri, putem converti aceste date în „*Dataset*”:

```
nodes = new vis.DataSet(edgesArray);
edges = new vis.DataSet(edgesArray);
```

Acum că avem două obiecte de tip „*DataSet*” în care menținem datele grafului, editarea acestuia este posibilă prin utilizarea metodei „*update()*” ce ne actualizează obiectele „*nodes*”, respectiv „*edges*”. Această metodă primește ca parametru, de asemenea, un vector de muchii sau noduri ce conțin modificările acestora. Spre exemplu:

```
nodes.update([{id:nodeId, color:newNodeColor}]);
edges.update([{id:edgeId, color:newEdgeColor}]);
```

Odată cu aceste actualizări putem vedea că graful nostru se modifică conform setărilor utilizatorului.

4.2.3 Desenarea grafului

Vis.js are o pagină de GitHub și, de asemenea, după cum am mai amintit în capitolul destinat tehnologiilor folosite, are și o pagină unde utilizatorii propun implementarea unor funcționalități noi pentru această bibliotecă ([23]). Aceasta, neavând nimic implementat legat de animația parcurgerii grafului, s-a propus acest subiect pentru viitoare versiuni [32]. Mai multe persoane s-au implicat în atingerea acestui scop și, astfel, s-a ajuns la implementarea unei funcționalități de animație a muchiilor, însă acestea nu au fost adăugate în bibliotecă, la momentul dezvoltării aplicației, așa că le-am adăugat într-un loc separat pentru a ne putea folosi de ele.

Ce s-a realizat este un modul ce are „*handler*”-e pentru „*pre*” și „*post*” animație. Această animație a fost de ajutor în procesul de desenare a grafului, însă nu de ajuns, deoarece constă doar în parcurgerea, cu ajutorul unu cerc, a muchiilor declarate dar nu și colorarea acestora și a nodurilor vizitate. Astfel am creat o funcție javascript de parcurgere a grafului urmată de colorarea căii folosind „*handler*”-ul de „*post*”-animare. Ca urmare, după fiecare parcurgere a muchiei, nodurile și muchiile vizitate își schimbă culoarea, marcându-le ca și traversate (cod sursă -Anexa 1).

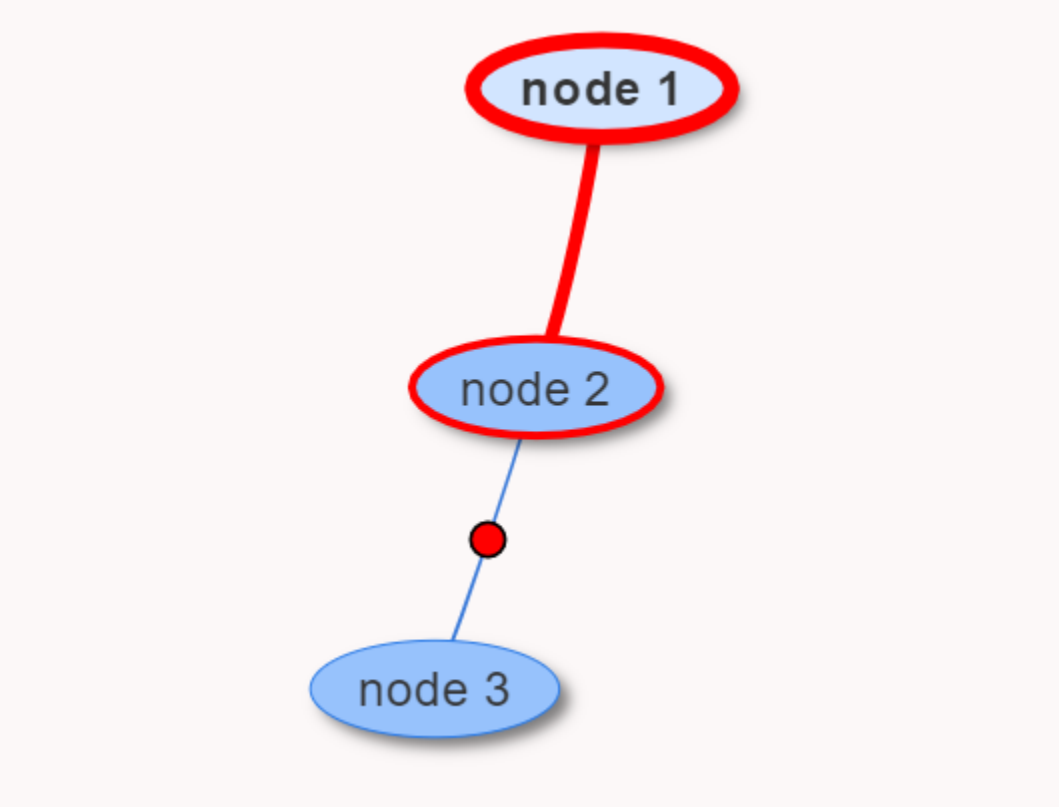

Fig. 4.4: Desenarea grafului în LearnGraphs

4.2.4 Implementarea algoritmilor

Pentru implementarea algoritmilor a fost nevoie de adaptarea lor în așa fel încât desenarea grafului sa fie corectă și completă, aceasta din urmă făcându-se pe baza unui vector de muchii. Muchiile unui graf sunt definite ca având obligatoriu următoarele proprietăți: „*id*”, „*from*”, „*to*”. Cu alte cuvinte vom ști tot timpul orientarea muchiei chiar dacă graful nu este orientat. Când avem un graf ca cel amintit mai devreme, parcurgerea muchiei se poate face în ambele sensuri, iar pentru a parcurge-o în sens opus cu metoda implementată de desenare, ii adăugăm ca și proprietate:

„*isBackward*”: „*true*”. Datorită faptului că graful poate fi orientat sau neorientat, aproximativ toți algoritmii implementați au doua feluri de parcurgere, respectiv colorare.

4.3 Manual de utilizare

În această secțiune vom vorbi despre funcționalitățile propriu-zise ale aplicației implementate și despre felul în care se pot utiliza acestea. Pentru acest lucru avem atașate capturi de ecran sugestive din aplicație.

4.3.1 Adăugarea nodurilor și a muchiilor

„**LearnGraphs**” are o pagină de start (Fig. 4.5) de unde putem naviga mai departe spre pagina funcționalităților. Aceasta este și una din componentele Angular 2 pe care le putem accesa prin intermediul procesului de „*routing*” amintit mai devreme.

Fig. 4.5: Pagină de start LearnGraphs

Apăsând pe butonul de „*Get Started*” aplicația ne trimite spre cealaltă componentă, cea în care se găsesc toate funcționalitățile. Prima pe care o vom discuta este de creare a nodurilor și adăugare de muchii, acest lucru constând în construirea, de fapt, a unui graf de către un utilizator. Biblioteca **Vis.js** ne pune la dispoziție un buton de editare (Fig. 4.6) ce duce la apariția a altor două dedicate nodurilor și respectiv muchiilor („*Add node*”, „*Add Edge*” – Fig. 4.7). Pentru a avea, din momentul accesării paginii, posibilitatea de a vedea aceste două butoane fără a mai trece printr-un alt pas, este nevoie adăugarea unei proprietăți pentru modulul „*manipulation*” din opțiuni, și anume: „*initiallyActive*”:„*true*”.

Crearea grafului, în această aplicație, constă, în primul rând, în adăugarea de noduri. Acest lucru se face prin apăsarea butonului, amintit și înainte, intitulat „*Add Node*”. Odată apăsat, în locul celor doua butoane apare un mesaj ce ne spune sa selectam un loc liber de pe tabla de lucru unde va fi poziționat nodul proaspăt creat. După ce am selectat zona, o mică fereastră cu anumite informații apare pe tabla (Fig. 4.8). Este vorba despre cele mai importante doua câmpuri ce descriu un nod, și anume: „*id*” și „*label*”. Locul unde apare „*id*”-ul nodului nu este editabil, în schimb este completat de către bibliotecă cu un număr generat la întâmplare („*random*”). Ce putem edita, însă, este „*label*”-ul redenumind nodul. Ce trebuie menționat este faptul că dacă numele nodurilor sunt identice, lucru nerecomandat, de altfel, în majoritatea cazurilor, nu este o problema, deoarece putem avea noduri cu același „*label*”. Pe de altă parte, „*id*"-urile nodurilor trebuie sa fie unice, lucru care se și întâmpla în aplicație, deoarece selecția lor și distingerea se face pe baza acestei proprietăți. În momentul în care utilizatorul este mulțumit cu datele minim necesare împărtășite la adăugarea nodului, acesta poate selecta butonul denumit „*OK*” în aplicație, astfel, creând în locul indicat un nou nod.

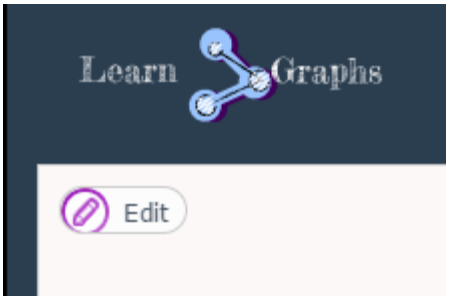

Fig. 4.6: Buton de editare LearnGraphs

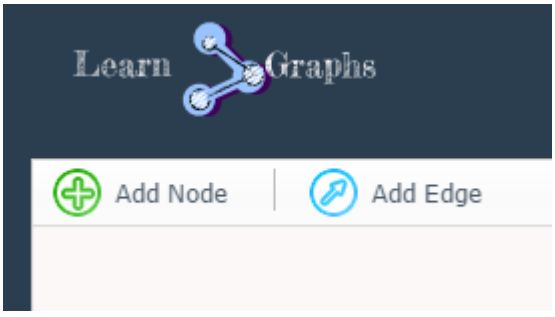

Fig. 4.7: Butoane de adăugare noduri și muchii LearnGraphs

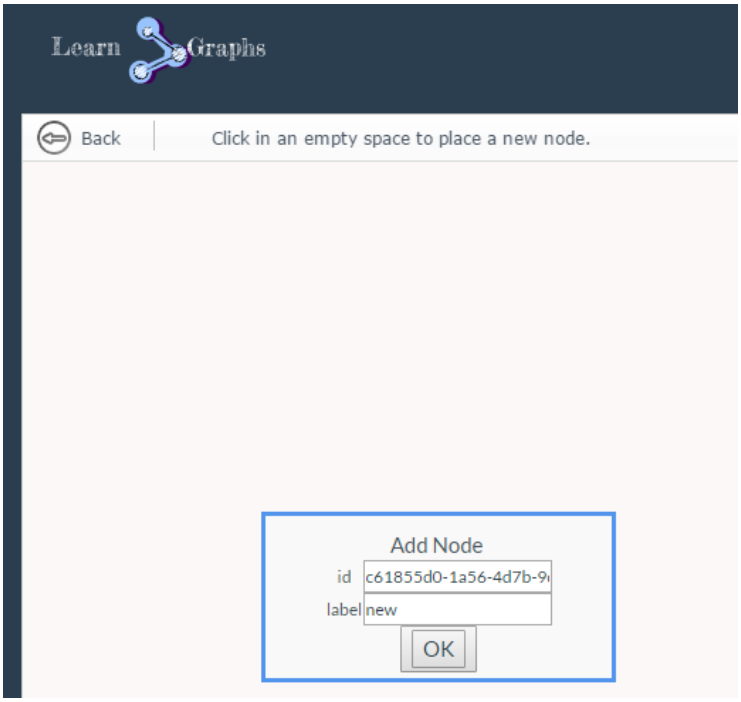

Fig. 4.8: Fereastră de informații nod în LearnGraphs

Cealaltă acțiune ce face opțiunea de creare a grafului completă este adăogarea de muchii. Acest lucru se poate realiza, de asemenea, prin apăsarea butonul „*AddEdge*” din interfața. Imediat după, cele două butoane dispar, la fel ca adăugarea nodurilor, însă de această dată în locul lor va apărea un mesaj specific pentru adăogarea de muchii. Acesta îndrumă utilizatorul spre următorul pas ce constă în folosirea cursorului, ținând click stânga apăsat pe nodul de la care dorește să pornească muchia, iar apoi trăgând de el până la nodul care va reprezenta capătul muchiei. Un „*dialog box*” va fi deschis instantaneu împreună cu un „*input*” în care ne cere sa introducem un titlu („*label*”) al muchiei. Se poate , de asemenea, sa nu se introducă nimic, rezultând ca muchia sa nu aibă titlu, iar în celălalt caz, muchia luând ca titlu ce a introdus utilizatorul la tastatura.

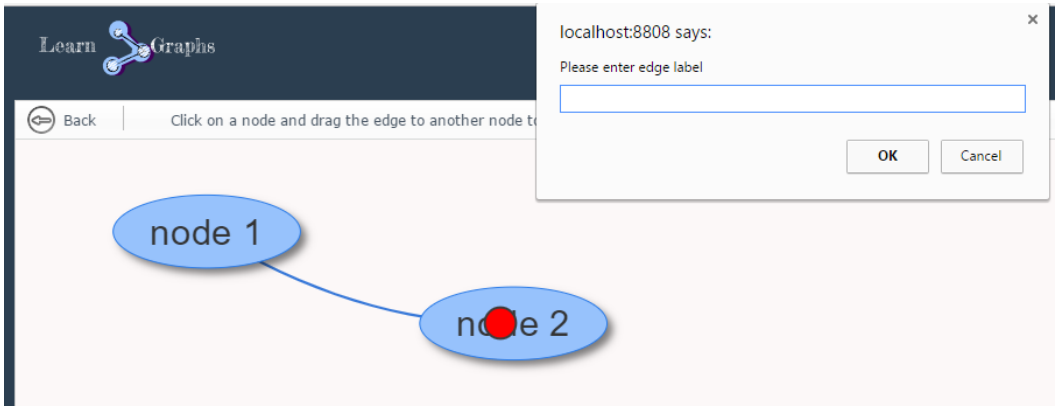

Fig. 4.9: Adăugarea unei muchii în LearnGraphs

Adăugarea unei muchii este posibilă și pentru muchii care au ca destinație și punct de plecare noduri identice. Daca acest lucru se întâmplă, un dialog apare pe pagină, înainte de cel de setare a titlului muchiei, și întreabă utilizatorul dacă aceasta este acțiunea pe care dorește să o execute.

În momentul în care utilizatorul crede ca nu mai are nevoie fie de un nod, fie de o muchie, acesta are posibilitatea de a le șterge. Ne putem folosi de această funcționalitate selectând elementul pe care dorim să-l ștergem și apoi apăsând butonul de ștergere („*Delete Selected*") din bara de deasupra tablei de lucru. Acest buton apare odată cu selectarea unui nod sau a unei muchii.

4.3.2 Exportarea și importarea grafurilor

Aplicația dispune de exportarea și importarea grafului, iar această funcționalitate este implementată după modelul furnizat de pe pagina oficială **Vis.js** [33]. Acest lucru este bazat pe plasarea unei arii de text (<*text-area*>) oriunde în aplicația noastră, reprezentând locul exportării grafului sub format JSON, respectiv al importării acestuia. Locul unde se întâmplă acest lucru este editabil, ceea ce reprezintă un mare avantaj pentru **LearnGraphs** deoarece oferă utilizatorilor opțiunea de a edita, a adaugă sau a șterge din textul grafului aflat în acest format.

În dreapta tablei de lucru cu grafuri am adăugat diferite butoane pentru fiecare funcționalitate în parte:

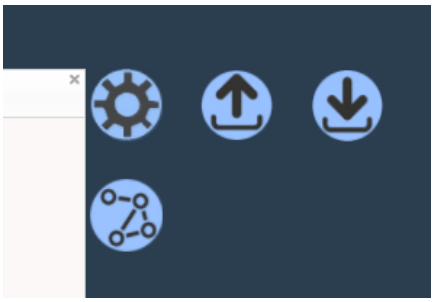

Fig: 4.10: Butoane funcționalități LearnGraphs

După cum putem observa, avem butoane diferite pentru exportare și importare. Fiecare dintre aceste butoane, odată apăsat, va deschide o bară laterală în interiorul căreia vom găsi elemente de care ne vom putea folosi.

Pentru exportarea sau importarea unui graf vom apăsa butonul cu săgeata în sus, respectiv butonul cu săgeata în jos. Pentru ca cele doua sunt identice, vom lua ca exemplu exportarea grafului. Așadar, vom apăsa butonul corespunzător și o bară laterală ce conține spațiul unde va fi exportat graful va apărea în partea dreaptă. Pentru importare, interfața este asemănătoare, mai puțin

butoanele specifice pentru acest lucru. Apăsând butonul „*Export*” putem observa că în locul destinat exportării apar niște date în format JSON ce reprezintă datele grafului curent.

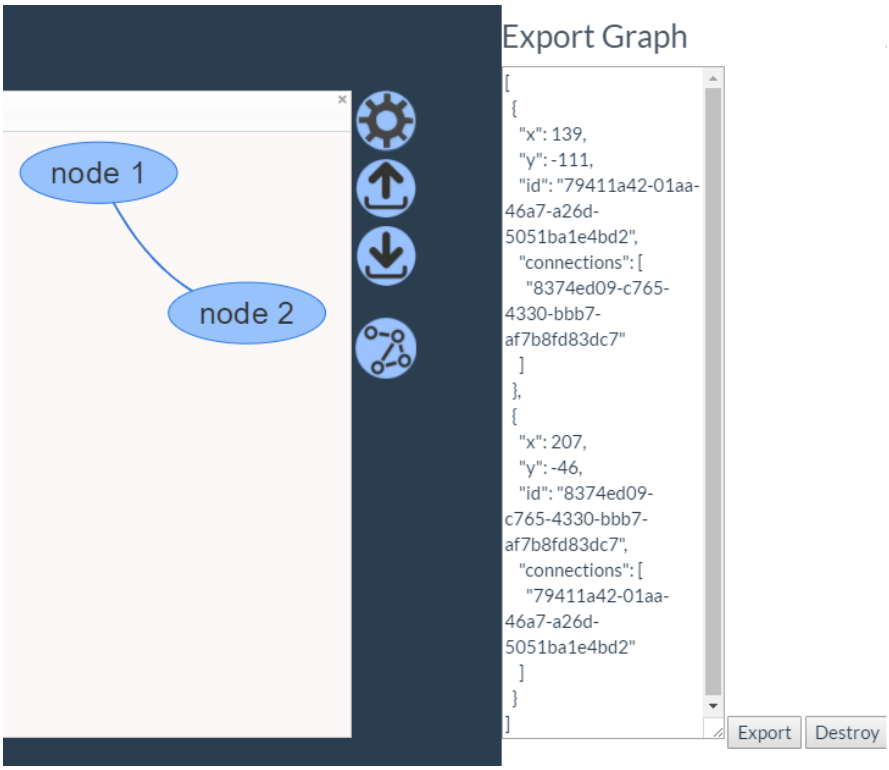

Fig. 4.11: Exportarea unui graf în LearnGraphs

Pentru distrugerea completă a grafului avem butonul denumit „*Destroy*”, prezent atât pentru exportare cât și pentru importare.

4.3.3 Alegerea setărilor generale pentru graf

După cum am menționat și în capitolul destinat tehnologiilor folosite, cu ajutorul bibliotecii **Vis.js** putem atașa grafului nostru un meniu de configurări. Aplicației **LearnGraphs** i-am selecționat câteva din opțiunile puse la dispoziție.

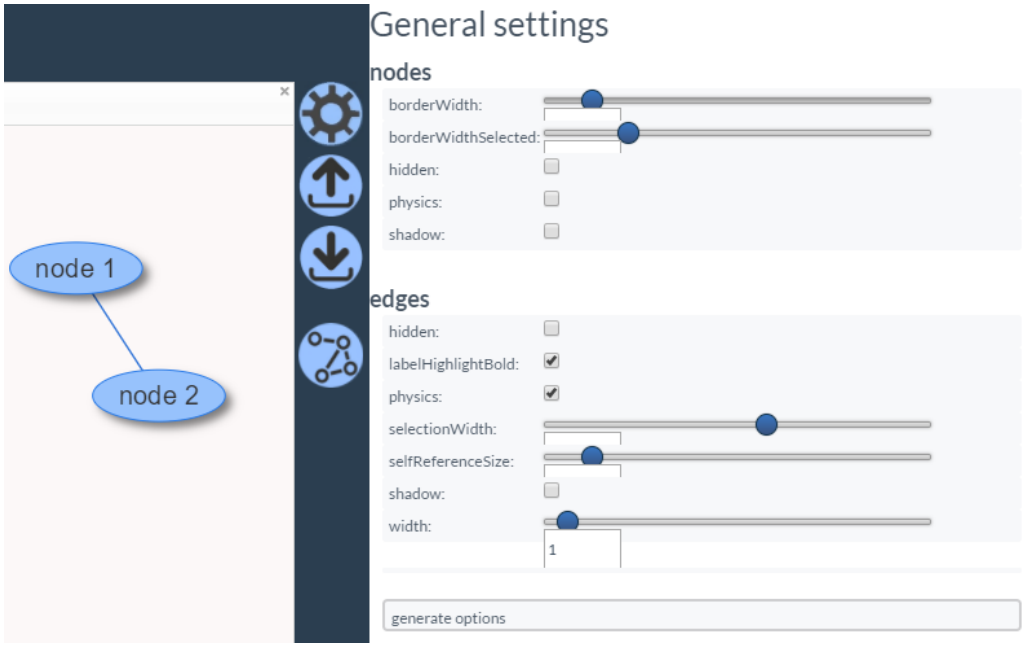

Fig. 4.12: Setări generale pentru graf în LearnGraphs

Când utilizatorul și-a configurat graful, acesta are posibilitatea de a genera opțiunile setate.

4.3.4 Editarea nodurilor și a muchiilor

Pentru a edita un nod sau o muchie nu trebuie decât ca utilizatorul sa selecteze acel element, după care să apese pe butonul „*Edit Node*” și respectiv „*Edit Edge*”. Acest buton apare, la fel ca cel de ștergere („*Delete selected*”) în momentul în care utilizatorul selectează un nod sau o muchie.

Ambele funcționalități, de editarea unui nod și de editare a unei muchii, declanșează afișarea unei bare laterale în partea dreapta a paginii. Aceasta conține opțiuni fie pentru noduri, fie pentru muchii, care daca sunt selectate, le atribuie nodului sau muchiei selectate.

Pentru noduri avem posibilitatea, în primul rând, de a le schimba numele sau „*label*”-ul cu ajutorul micii ferestre de informații asemănătoare cu cea de la adăugare. Această fereastră apare odată cu bara laterală, la apăsarea butonului „*Edit Node*”. Altă posibilitate în editarea nodurilor este cea de a le schimba forma prin selectarea ei, având la dispoziție o serie de figuri geometrice. Și nu în ultimul rând putem să le schimbăm culoarea, alegând-o prin intermediul unui selector de culoare pus la dispoziție („*color picker*”).

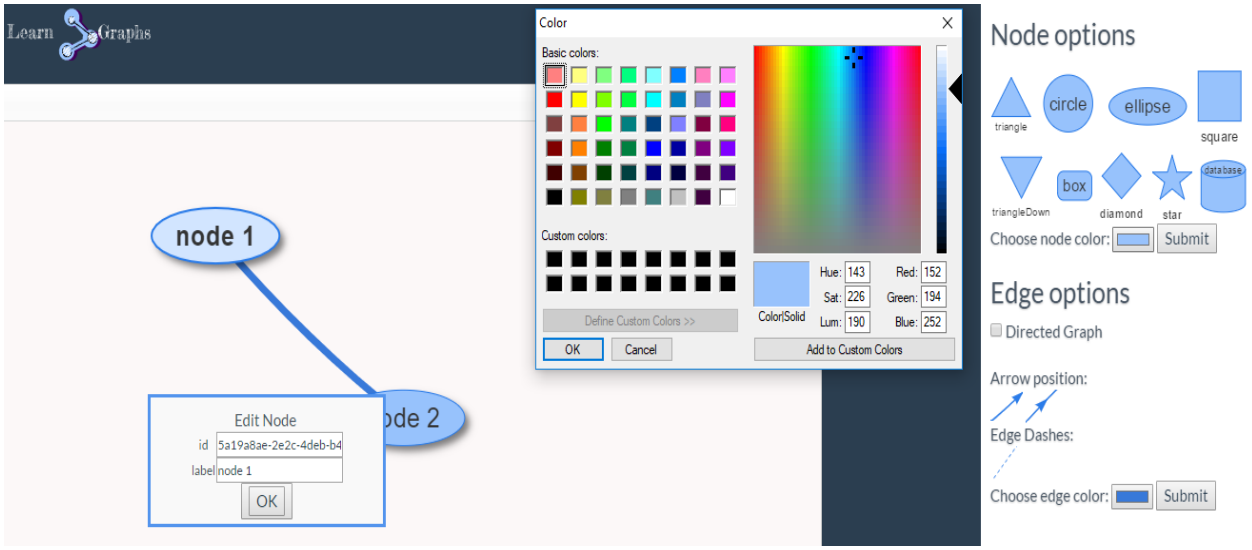

Fig. 4.13: Editarea nodului în LearnGraphs

Când editǎm o muchie, avem două posibilități: ori ii schimbăm nodurile pe care le leagă, ori ii schimbăm înfățișarea. Pentru prima opțiune, capetele muchiei devin două puncte mari roșii ce pot fi mutate de la un nod la altul. Dacă cele două puncte indică spre același nod, atunci aplicația va trata cazul exact ca la adăogarea unei muchii. Asta înseamnă că un „*dialog box*” ce ne cere să confirmăm acțiunea noastră va apărea.

În ceea ce privește cea de-a doua posibilitate, de modelare a muchiei, aici am integrat și funcționalitatea de transformare a grafului neorientat într-unul orientat și invers. Acest lucru se face prin bifarea căsuței în dreptul căreia scrie „*Directed Graph*”, și determină posibilitatea folosirii unor alte modelari ale muchiei, și anume: punerea săgeții unei muchii orientate în capăt sau în mijloc. O muchie o mai putem face punctată, respectiv putem să-i modificăm culoarea exact ca la noduri.

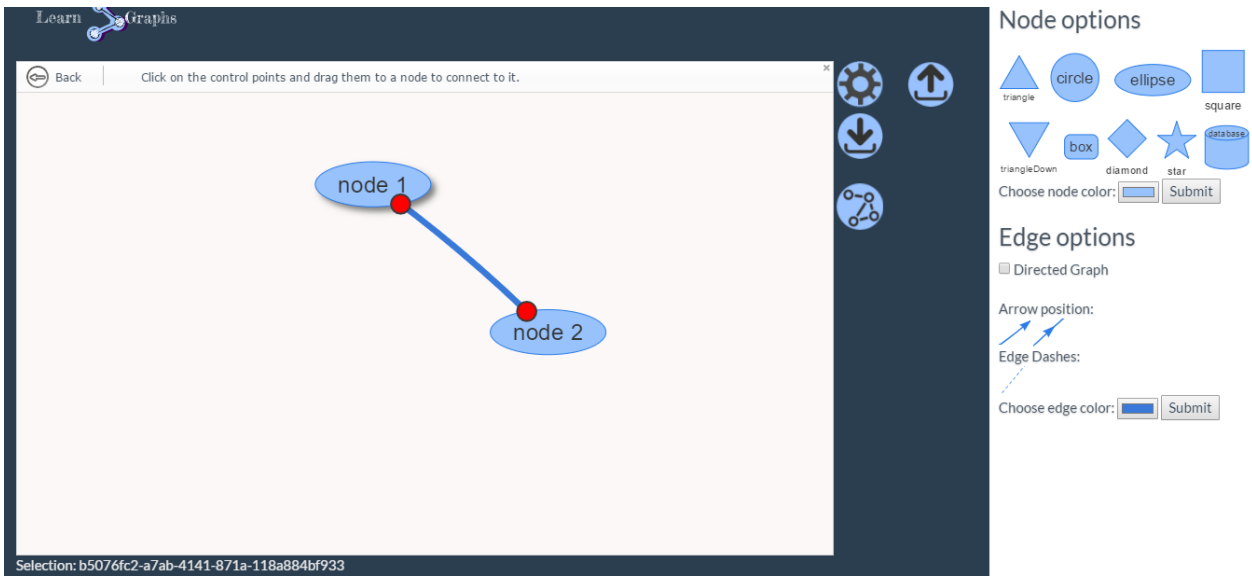

Fig. 4.14: Editarea unei muchii în LearnGraphs

4.3.5 Rularea unor algoritmi pe grafurile create sau importate

Pentru această secțiune vom lua ca exemplu un algoritm aplicabil unui graf, pentru a-l avea ca exemplu. Mai întâi vom crea un graf după procesul pe care l-am descris anterior. Îl putem modela, de asemenea, pentru că asta nu va încurca cu nimic efectuarea algoritmului.

În așa fel încât să accesăm algoritmii disponibili în **LearnGraphs** vom apăsa butonul cu un mic graf desenat pe el. Aici vom putea observa o înșiruire de butoane ce reprezintă algoritmii disponibili. Pentru a da ca exemplu, vom efectua algoritmul BFS (Breadth First Parsing) pe un graf creat de noi. Pentru acest lucru vom avea nevoie de un nod de la care să înceapă parcurgerea.

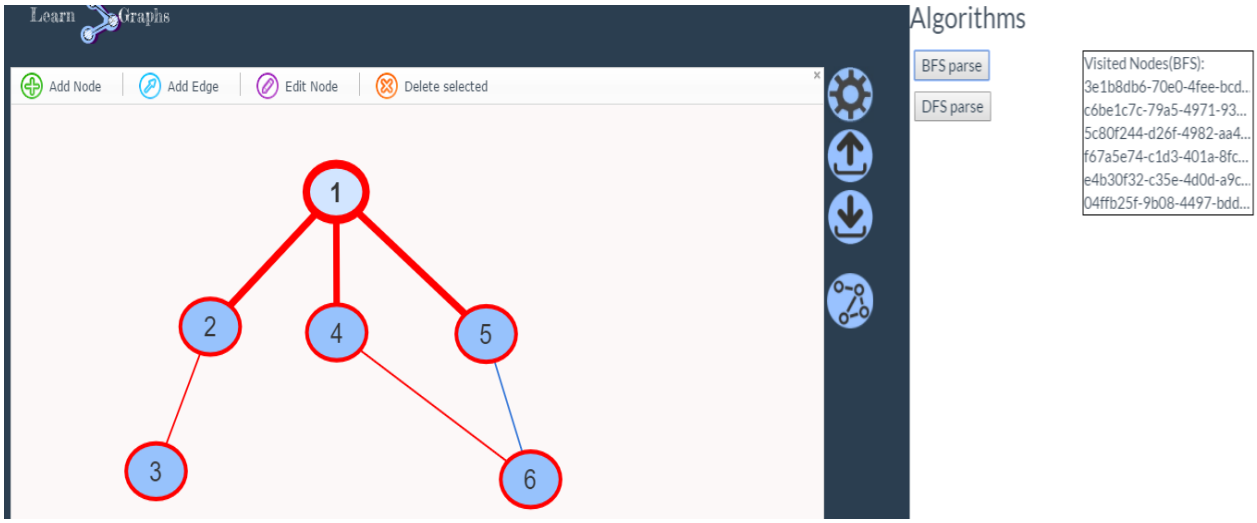

Fig. 4.15: Parcurgerea grafului cu algoritmul BFS în LearnGraphs

După cum putem observa, avem mai sus parcurgerea unui graf în lățime, iar în dreapta, în bara laterală, se află ordinea nodurilor parcurse cu acest algoritm, fiind afișate „*id*”-urile fiecăruia.

Capitolul 5

Concluzii

Prin această lucrare de licență am reușit sa creăm o aplicație ce execută algoritmi importanți din teoria grafurilor într-un mod vizual. În urma executării fiecărui algoritm în parte rezultă o serie de informații utile, spre exemplu ordinea nodurilor traversate. „**LearnGraphs**” este o aplicație novatoare atât din punct de vedere al tehnologiilor folosite, aceasta folosind una dintre cele mai utilizate „*framework*”-uri la ora actuală pe partea de „*front-end*”, Angular 2, cât și din punct de vedere al funcționalităților.

Am reușit sa creăm o aplicație al cărei conținut se încarcă o singură dată, accesând-o prin intermediul unui navigator de internet. Biblioteca „*Vis.js*” a fost de mare ajutor, de asemenea, prin modul acesteia de a desena grafuri. Folosindu-mă de acest lucru am creat o interfața ce ne ajuta în procesul de modelare și modificare a grafului într-un mod dinamic. Așadar, vom putea modela fiecare nod în parte a grafului desenat, dar și fiecare muchie.

O funcționalitate foarte utilă în aplicație este posibilitatea de exportare și respectiv importare a unui graf. Daca vrem să testăm un algoritm pe un graf anume îl putem fie crea, fie importa. Importarea este o metodă rapidă și este disponibilă și în aplicația noastră. De asemenea prima dată putem să-l creăm cu uneltele disponibile pentru desenare, după care îl putem exporta și salva într-un fișier text în formatul JSON. Astfel, data viitoare când vom avea nevoie de același graf, nu vom face decât sa-l importăm. Funcționalitatea principală, de rulare a unor algoritmi pe grafurile desenate, poate fi dezvoltată implementând algoritmi mai complicați din timpul facultății, gama de utilizatori lărgindu-se și spre studenți și profesori universitari.

Din punct de vedere al codului sursă al aplicației, acesta se poate îmbunătăți pe viitor, integrând anumite elemente din interfață în componente Angular 2, ca urmare, având o mai bună organizare a acestuia. Astfel, procesul de dezvoltare a aplicației va fi accelerat, iar atunci putând fi adăugate funcționalități noi.

Bibliografie

[1] - Cristea, Ioniță, Pistol, *Inteligență Artificială*

[2] - Mariana Miloșescu, *Informatică Intensiv XI*, Editura Didactică și Pedagogică, R.A., An 2006

[3] - *Garmin | What is GPS?*, http://www8.garmin.com/aboutGPS/

[4] - *10 best GPS app and navigation app*, http://www.androidauthority.com/best-gps-app-and-navigation-app-for-android-357870/

[5] - *UML Diagram Types With Examples for Each Type of UML Diagrams*, http://creately.com/blog/diagrams/uml-diagram-types-examples/

[6] - *ArgoUML*, http://argouml.tigris.org/

[7] - *Graph Theory | University of Southampton*, http://www.southampton.ac.uk/courses/modules/math3033.page

[8] - *Graph Creator*, http://illuminations.nctm.org/Activity.aspx?id=3550

[9] - *Graph Online*, http://graphonline.ru/en/

[10] - *GoJS Diagrams*, https://gojs.net/latest/index.html

[11] - *GoJS API*, https://gojs.net/latest/api/index.html

[12] - *GoJS Samples*, https://gojs.net/latest/samples/index.html

[13] - *Alchemy.js*, http://graphalchemist.github.io/Alchemy/#/

[14] - *Alchemy.js Full Application*, http://graphalchemist.github.io/Alchemy/#/examples/Full_Application/Viz

[15] - *Cytoscape.js*, http://js.cytoscape.org/

[16] - *The R Project for Statistical Computing*, https://www.r-project.org/

[17] - *Bioconductor – Rcyjs*, http://www.bioconductor.org/packages/release/bioc/html/RCyjs.html

[18] - *Angular 2 and Typescript – A High Level Overview*, https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview

[19] - *Angular 2 Tutorial*, https://www.tutorialspoint.com/angular2/

[20] - *vis.js - A dynamic, browser based visualization library*, http://visjs.org/

[21] - *Almende – Self-organizing Networks – Almende B.V.*, http://www.almende.com/home

[22] - *almende - GitHub*, https://github.com/almende

[23] - *Issues – almende/vis GitHub*, https://github.com/almende/vis/issues?q=is%3Aopen+is%3Aissue+label%3AFeature-Request

[24] - *vis.js – Manipulation documentation*, http://visjs.org/docs/network/manipulation.html

[25] - *vis.js – Modules*, http://visjs.org/docs/network/#modules

[26] - *vis.js – Methods*, http://visjs.org/docs/network/#methods

[27] - *vis.js – Events*, http://visjs.org/docs/network/#Events

[28] - *ng-sidebar*, https://www.npmjs.com/package/ng-sidebar

[29] - *ng-sidebar Demo*, https://echeung.me/ng-sidebar/

[30] - *Creately*, https://creately.com/app/?tempID=h165rwt81&login_type=demo#

[31] - *Draw.io*, https://www.draw.io/

[32] - *Add animation to the edges issue*, https://github.com/almende/vis/issues/507

[33] - *vis.js - Saving and loading networks*, http://visjs.org/examples/network/other/saveAndLoad.html

[34] – *Graph Theory*, http://www.personal.kent.edu/~rmuhamma/GraphTheory/MyGraphTheory/defEx.htm

Anexe

Anexa 1

Funcția de desenare a unui graf

```
var sequentialEdgesToAnimate = [
      {edge:edgeId1}
      {edge:edgeId2}
      {edge:edgeId3, isBackward:true}
];

function parseGraph(startingEdgeNum) {
      var currentEdge = edgesArray.filter(function (el) {
            return el.id === sequentialEdgesToAnimate [startingEdgeNum].edge;
      });

      var isBackward = sequentialEdgesToAnimate [startingEdgeNum].isBackward;

      var fromNode = nodesArray.filter(function (el) {
            return el.id === currentEdge[0].from;
      });
```

```
var toNode = nodesArray.filter(function (el) {
      return el.id === currentEdge[0].to;
});

if(!isBackward){
      visitNode(currentEdge[0].from);
} else {
visitNode(currentEdge[0].to);
}

graph.animateTraffic(
      /* first edge to start animating */
      sequentialEdgesToAnimate [startingEdgeNum] ,
      /* onPreAnimationHandler*/
      null,
      // /*onPreAnimateFrameHandler*/
      null ,
      // /*onPostAnimateFrameHandler*/
      null ,

      /* onPostAnimationHandler */
      function(edgesTrafficList) {
      var currentEdgeId =
                  sequentialEdgesToAnimate [startingEdgeNum].edge;

      edges.update([{id:currentEdgeId, color:'#FF0000'}]);

      if(!isBackward){
            visitNode(currentEdge[0].to);
      } else {
            visitNode(currentEdge[0].from);
      }
```

```
            startingEdgeNum++;
            if (startingEdgeNum < sequentialEdgesToAnimate.length) {
                  parseGraph(startingEdgeNum);
            }
      });
}
```