Cuprins

1. Introducere

1.1. Motivaţie

Odată cu intrarea în lumea informaticii, o persoană are de învăţat anumiţi algoritmi sau structuri de date care îi vor fixa o bază şi îl vor ajuta mai departe pentru rezolvarea problemelor pe care le întâmpină. De obicei, premisa care stă la baza învaţării acestora este aceea că trebuie înţeles mai întâi modul de rulare a programului (funcţii, bucle, recursivitate). Apoi, algoritmii şi structurile de date pot fi abstractizate pentru o înţelegere mai uşoară a unui program, prin faptul că nu mai suntem restrictionaţi de implementare ci mai mult de ideile principale. De aceea, în primă fază, se foloseşte un pseudocod pentru a explica un algoritm sau o structură de date.

Abstractizarea ne îndepărtează de lucrurile cu care suntem obişnuiţi (de exemplu, lucrul cu grupurile algebrice) şi ne pune câteodată în situaţii inconfortabile, în care ne este greu să vizualizăm/imaginăm soluţia unei probleme. Astfel, e mai uşor să facem gradual această tranziţie, prin exemple sau prezentarea unor cazuri mai simple.

1.2. Obiectiv

Această aplicaţie are scopul de a uşura şi minimiza timpul de învăţare a unui algoritm sau unei structuri de date descrise în următoarele pagini. Există situaţii când, chiar dacă codul scris pare corect, e mai uşor de vizualizat prin imagini şi rulare pas cu pas a programului. Diferenţa dintre modul în care sunt ţinute datele în calculator şi semnificaţia pe care le-o dăm sau modul în care vizualizăm pentru a uşura înţelegerea, constituie unul din principalele scopuri cu care a fost scrisă aplicaţia.

Totodată, se pune la dispoziţia utilizatorului un mediu de lucru flexibil, controale de flux ale programului (butoane de începere/pauză, un *timer* pentru fiecare pas al programului), o componentă de vizualizare a acelui algoritm sau structură de date, puncte de merit pentru răspunsuri corecte, pentru transforma învăţarea într-o experienţă plăcută şi provocatoare.

Obiectivul final este ca un utilizator să poată învăţa în modul ales de el, mai încet sau mai rapid, şi energia depusă să fie cât mai mare pe acel element şi nu pe dificultatea utilizării acestei aplicaţii.

1.3. Soluţii existente

Există mai multe soluţii de acest gen, gratuite sau nu, iar dintre acestea am ales să discut şi compar trei. Când avem în vedere o soluţie pentru problema de învăţare a unui algoritm sau structură de date, luăm în considerare următoarele aspecte:

- **flexibilitatea aplicaţiei** (dacă utilizatorul poate modifica intrările, schimba codul, pune *breakpoints* sau comentarii);
- **tip de vizualizare** (dacă se poate da *zoom*, dacă are culori, dacă interfaţa este statică sau interactivă);
- **controlul de flux** (dacă putem opri programul la un anumit pas, inversa pasul curent, stabili un interval de parcurgere a liniilor de cod);

Toate aceste caracteristici contează pentru un utilizator şi pentru modul de învăţare pe care îl aplică.

1.3.1. *PathFinding.js*

Aceasta este o aplicaţie [1] care simulează diferiţi algoritmi de găsire a drumurilor minime pe o matrice. Totuşi, unii din algoritmii prezentaţi pot fi aplicaţi şi pe grafuri. Câţiva dintre algoritmii care pot fi selectaţi de utilizator sunt:

- *A**
- *Dijkstra*
- *Breadth-First search*
- *Jump-Point search*

Utilizatorul poate alege pe oricare dintre aceştia şi tipul de distanţă care se foloseşte la calcularea costului unui drum (euclidiană, Manhattan). Pe lângă acestea, utilizatorul dispune de o vizualizare interactivă care acceptă comenzi de *mouse-click* sau *drag-and-drop*. Există o matrice de dimensiuni fixe, cu un punct de start reprezentat prin verde închis şi un punct de oprire reprezentat prin roşu.

Mai mult, comanda de *click* pe o celulă comută o celulă liberă într-una invalidă, şi invers.

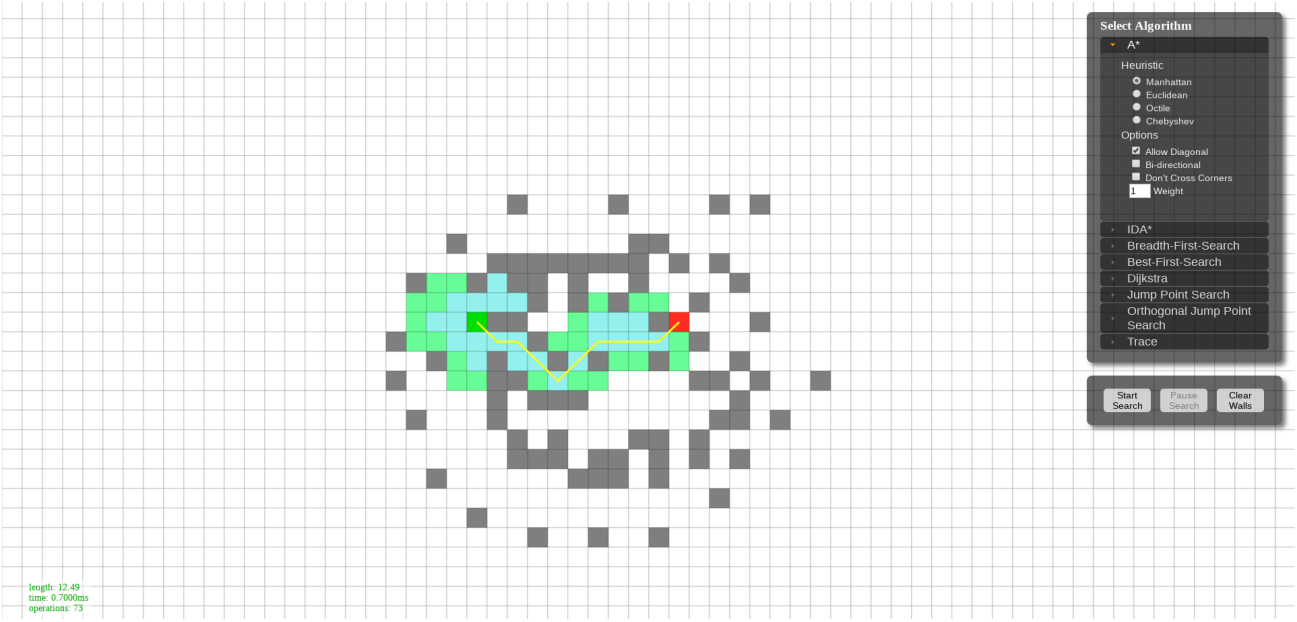

Figura 1-1 *PathFinding.js*

Aplicația are butoane de începere și oprire a algoritmului, dar rulează foarte rapid pentru o matrice cu dimensiunile ilustrate, astfel că utilizatorul nu are timp să apese butonul de oprire. Totuși, oferă o funcționalitate folositoare în unele situații.

Astfel, aplicația este flexibilă, utilizatorul putând schimba aproape tot cu excepția dimensiunilor matricei. Ea oferă o vizualizare care diferențiază diferitele tipuri de celule și care este interactivă (acțiunile facându-se cu ajutorul *mouse*-ului). Totuși partea de control al fluxului oferă un set incomplet, lipsând controlul de timp și altele.

1.3.2. *USF Data Structure Vizualizations*

Universitatea din San Francisco, departamentul de Computer Science, a construit o listă de vizualizări pentru diverși algoritmi și structuri de date [2]:

- sortări (*Radix sort*, *Bucket sort*, *Heap sort*);
- structuri de indexare (*Hash tables*, arbori indexați binar, arbori *AVL*);
- algoritmi de grafuri (*Depth-First search*, *Breadth-First search*, componente conexe, arbore parțial de cost minim);
- programare dinamică (numere Fibonacci, problema celui mai lung subșir comun);

Având o mulţime vastă de algortimi şi structuri de date, ne vom referi la principalele componente care se găsesc în vizualizări şi la avantajele/dezavantajele pe care această aplicaţie le are.

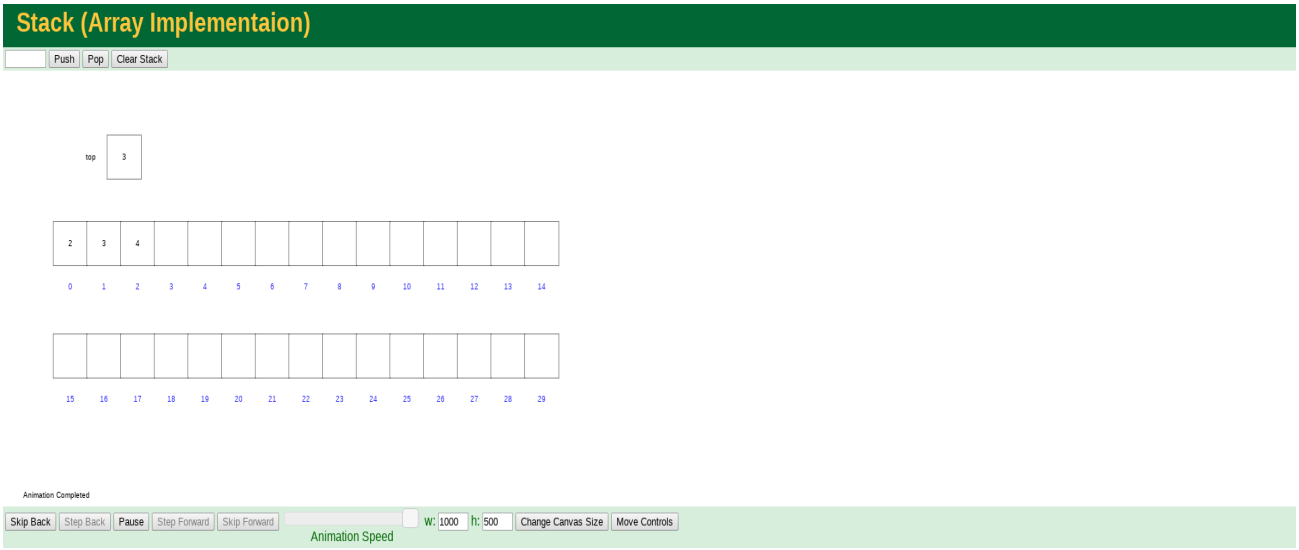

Figura 1-2 *USF - Stack implementation*

Ca şi controale de flux, aplicaţia oferă:

- un *slider* pentru viteza animaţiei unei operaţii din cadrul exerciţiului;
- butoane de începere şi pauză;
- butoane care îţi oferă posibilitatea să mergi doar câte un pas în algoritm, înainte sau înapoi;
- butoane speciale care depind de exerciţiu şi definesc operaţiile pe care un utilizator le poate selecta;

Majoritatea vizualizărilor nu sunt interactive, ci doar operaţiile specifice acelui exerciţiu schimbă vizualizarea. Totuşi, aplicaţia are animaţii destul de interesante şi diverse, plăcute utilizatorului. Vizualizarea este conţinută într-un bloc *canvas* care se poate redimensiona prin editarea a două valori, *width* si *height.*

Aplicaţia este relativ flexibilă, putând primi ca date de intrare şi ieşire orice valori în formatul descris de exerciţiu. Un dezavantaj este faptul că nu se pot introduce mai multe operaţii deodată. Astfel, sesiunea a expirat, trebuie reintroduse operaţiile, de mână, din nou. De obicei, există

mai multe operații care se fac asupra unei structuri de date sau algoritm, și ar fi natural să se poată preciza numărul de operații pe prima linie a datelor de intrare, iar după aceea să se descrie fiecare operație pe următoarele linii.

În concluzie, aplicația dezvoltată de Universitatea din San Francisco oferă o gamă largă de algoritmi și structuri de date și o pagină de simulare destul de completă. Cu toate acestea, există câteva dezavantaje, cum ar fi vizualizările care nu pot fi redimensionate cu ajutorul acțiunilor *mouse*-ului, sau operațiile care trebuie introduse una câte una.

1.3.2. *CsAcademy Applications*

Acest *site* [3] este folosit în principal pentru concursuri de algoritmică, dar are niste sub-aplicații pe care orice utilizator le poate folosi în cazul în care dorește să vizualizeze o structură de date.

Cele mai importante aplicații sunt *„Graph Editor”* și *„Geometry widget”*. Aceste aplicații construiesc o vizualizare bazată pe date de intrare scrise de utilizator.

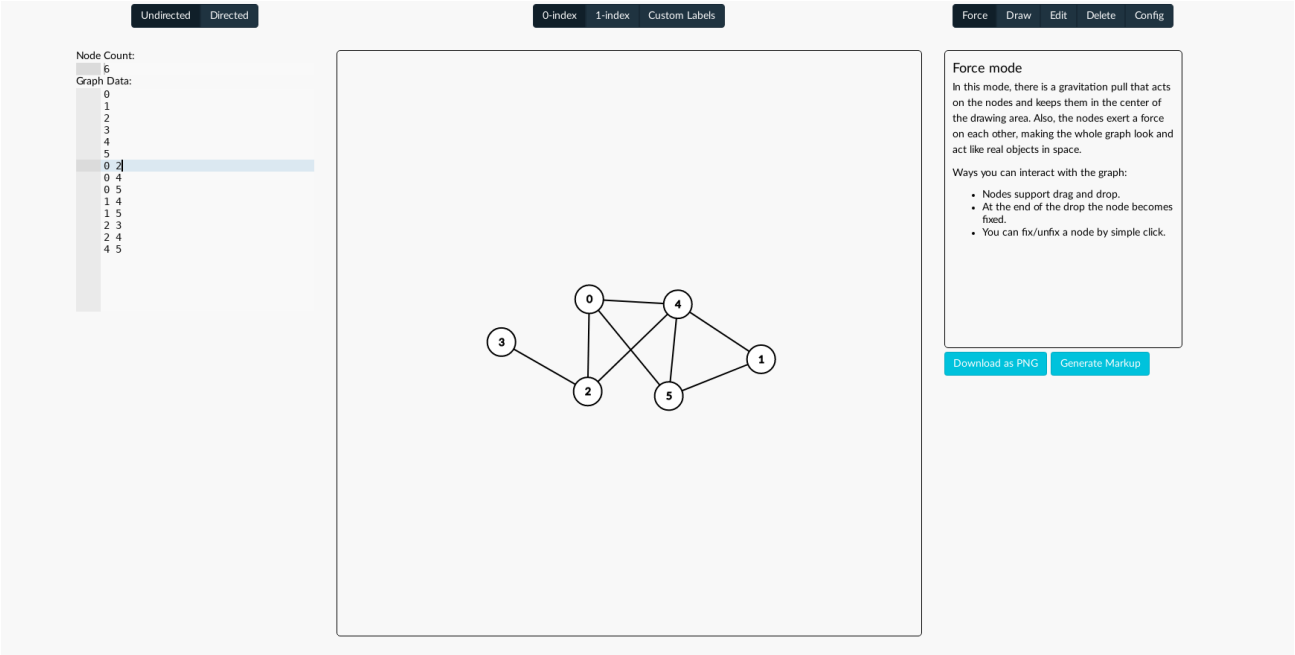

Figura 1-3 *CsAcademy - Graph Editor*

Astfel, la partea de control al fluxului aplicațiile nu au nimic utlizabil. În schimb, vizualizările sunt interactive și pot fi declanșate prin utilizarea *mouse*-ului (*scroll*, *drag-and-drop*).

Mai mult, datele de intrare pot fi editate în totalitate de utilizator, fiind restricționate de formatul impus de aplicație. Totodată, există mai multe moduri pentru editarea vizualizării fară ajutorului datelor de intrare (*Draw*, *Edit*, *Delete*).

În concluzie, aplicațiile nu oferă o gamă largă din care utilizatorul să poată alege și nici o simulare a unui program. Vizualizările sunt singura categoria unde aplicațiile oferă functionalități, chiar mai mult față de celelalte două aplicații descrise mai sus.

3. Tehnologii utilizate

3.1. *Javascript*

Fiind o aplicație web, limbajul principal folosit, pe lângă *HTML* și *CSS*, este *Javascript*.

Javascript este un limbaj de programare orientat obiect bazat pe conceptul prototipurilor. Totodată, este un limbaj dinamic, având urmatoarele caracteristici:

- obiectele pot dobândi noi proprietăți și funcții la *run-time*;
- se pot crea/rula *script*-uri prin ajutorul funcției *eval()*;
- *object introspection* (se poate examina tipul de date sau proprităţile unui obiect la *run-time*);

Astfel, este un limbaj flexibil și este ideal pentru o aplicație unde nu se cunosc datele cu care se va lucra dinainte (fiind o aplicație de tip *client-server*).

Javascript este asemănător limbajelor *C++* sau *Java*, având funcții, obiecte, tipuri de date, dar diferă prin alte module cum ar fi obiectul *document* care este folosit pentru a comunica cu elementele de pe pagina ce formează DOM (*Document Object Model*).

În istoria web-ului, la început *site*-urile erau statice, utilizatorul având posibilitatea să le vizualizeze dar nu interacționa. Mai târziu, a fost introdus *Javascript* cu scopul de a combate acest dezavantaj.

Javascript rulează pe partea de client a unei aplicații web (în *browser*), care poate fi folosit pentru a programa comportamenul aplicației în cazul unui eveniment creat de utilizator (de exemplu, acțiuni provocate de *mouse* sau tastatură). Mai mult, prin ajutorul *browser*-ului, poate comunica cu

alte calculatoare cu scopul trasnferului de informații, prin ajutorul apelurilor *HTTP* (*Hypertext Transfer Protocol*).

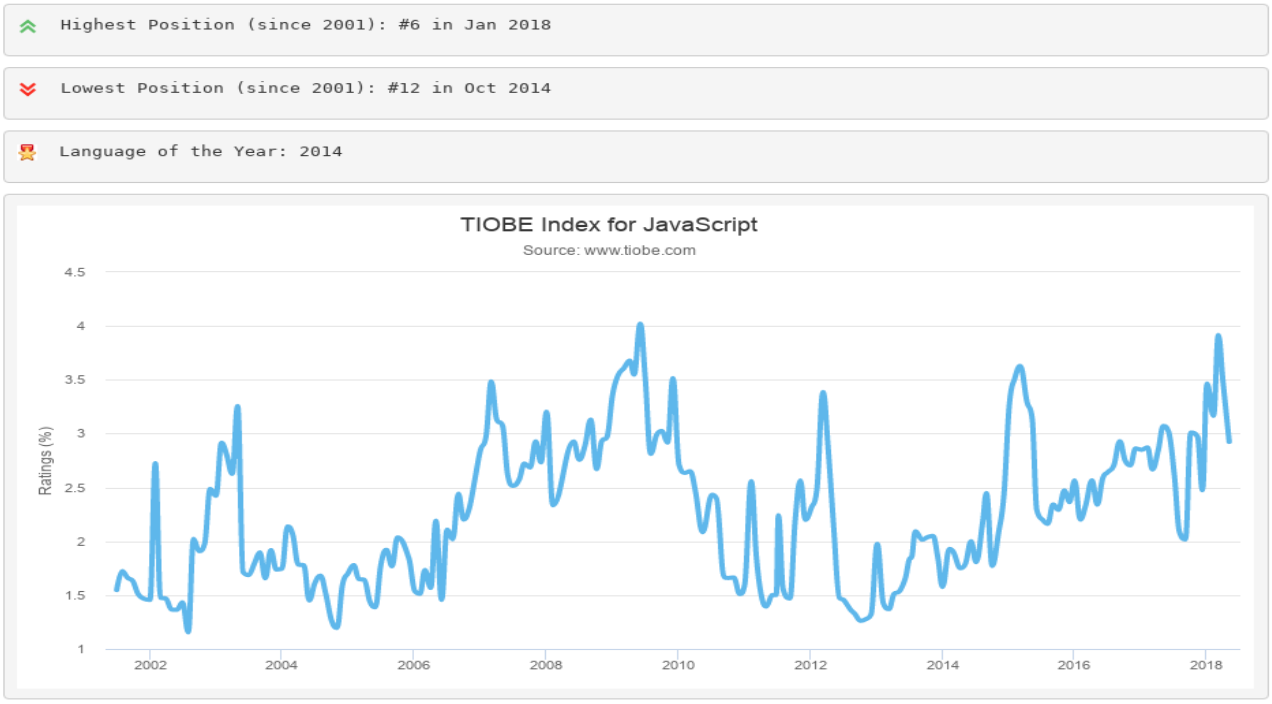

Figura 3-1 *Javascript TIOBE index*

Javascript, în combinație cu *HTML*/*CSS* și mediul oferit de un browser (*cookies*, *browser storage*), formează un grup matur și puternic de tehnologii, comunitatea fiind vastă și de ajutor.

Se observă din graficul obținut de cei de la *TIOBE* că *Javascript*, chiar și după douăzeci de ani de folosire în lumea dezvoltării aplicațiilor web, rămâne în top zece cele mai folosite limbaje de programare de-a lungul globului.

3.2. *Angular*

Angular [4] este succesorul popularului framework ***AngularJs*.**

AngularJs este un framework bazat pe Javascript, menținut în principal de Google, și permite dezvoltarea de *single-page applications.* Principalele obiective de design sunt:

- să decupleze manipularea *DOM*-ului (*Document Object Model*) față de logica aplicației;
- să decupleze *frontend*-ul de *backend* (are rolul de a împărți sarcinile separat, acceptând ideea lucrului în paralel);

- să furnizeze o structură coerentă pe masură ce se vor dezvolta diferitele module ale aplicației (UI, logica de business, testarea);

AngularJs implementează *pattern*-ul *MVC* (*Model-view-controller*) pentru a separa prezentarea, datele și logica componentelor. Folosindu-se de *DI* (*Dependency Injection*), aduce tradiționalele servicii *server-side*, cum ar fi *controller*-ele dependente de *view*-ul utilizatorului, în *client-side*. Astfel, reduce munca depusă de server de a injecta serviciile necesare clientului la un anumit moment.

Totodată, *AngularJs* citește codul HTML care formează pagina (*view*) care poate conține *tag*-uri atribut specifice. Acestea sunt interpretate ca directive pentru a forma un *binding* între elementele de *input* și *output* ale paginii și un model reprezentat de un cod scris în *Javascript* standard. Acest model aparține unui *controller*, o funcție scrisă cu ajutorul bibliotecii oferite de *AngularJs*.

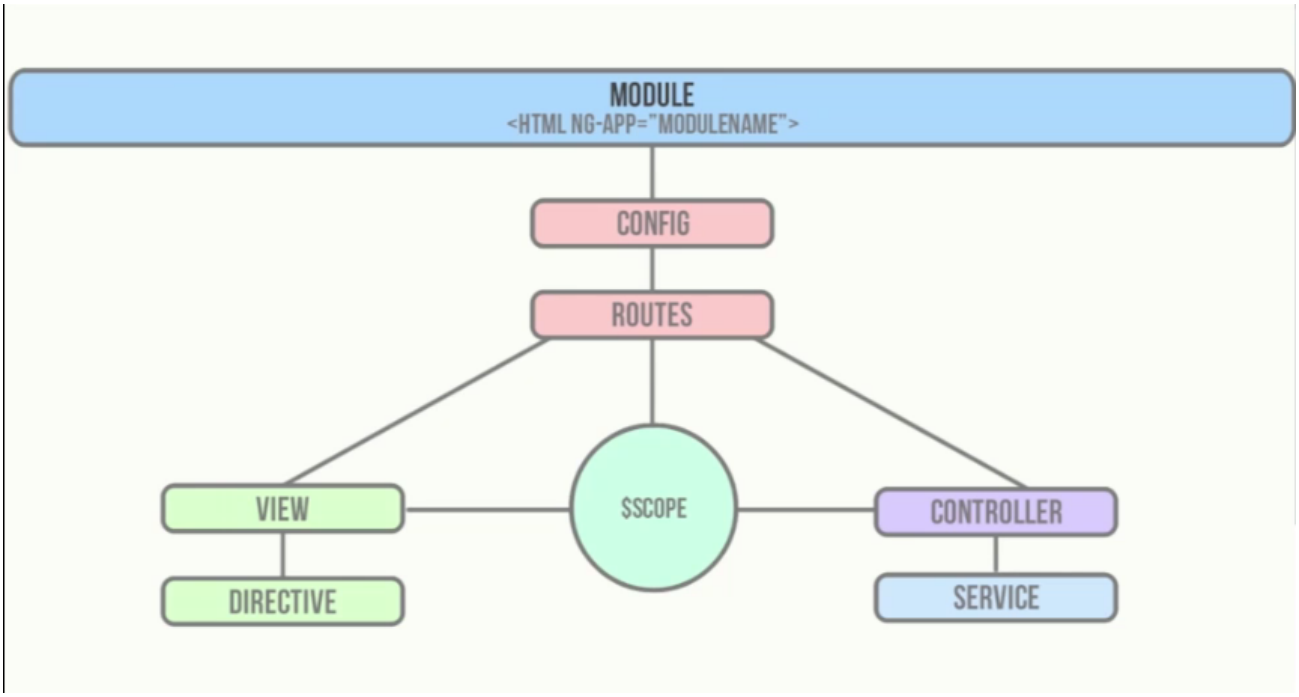

Figura 3-2 *AngularJs structure*

Directivele menționate mai sus au rolul de a extinde codul HTML și a ușura interacțiunea dintre *view* și *controller*. *AngularJs* oferă multe directive pe care un programator le poate folosi, unele dintre acestea fiind:

- *ng-app* (inițializeaza componenta de bază care este denumita *root component*);

- *ng-controller* (specifică o clasă de tip *controller* care ştie sa evalueze expresii HTML);
- *ng-model* (stabileşte un *two-way data-binding* între elementul HTML şi variabila model);
- *ng-repeat* (instanţiază câte un element pentru fiecare *item* din listă care poate fi folosit în *view*);

O altă structurare care o aduce *AngularJs view*-ului este *scope tree*. Iniţializând elementele HTML prin directiva *ng-controller*, se creează un element numit *$scope*, care este intermediarul comunicării dintre *view* şi *controller*.

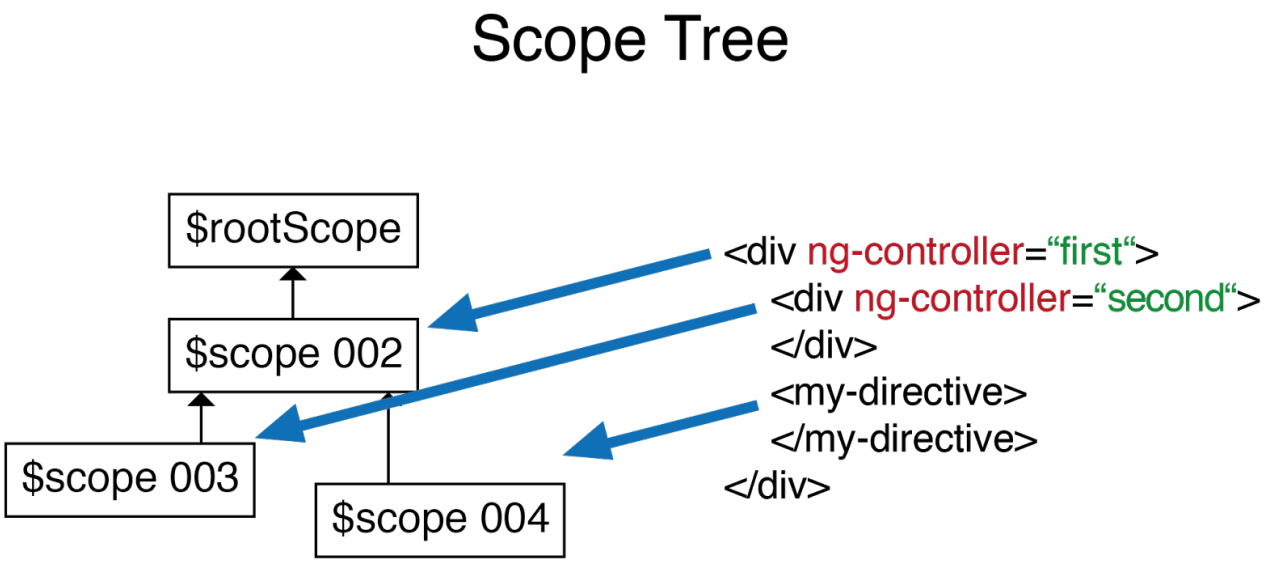

Figura 3-3 *AngularJs scope tree*

Acest arbore de scopuri se comportă asemănător moştenirii în contextul limbajelor orientat obiect. Dacă un element căruia i s-a atribuit un scop este un descendent ca şi element DOM al altui element cu un scop diferit, scopul descendent moşteneşte modelul din scopul strămoş.

Angular se bazează pe caracteristicile şi flexibilitatea oferita de *AngularJs*, oferind un pachet complet (*routing*, componente, module, *dependency injection*) şi posibilitatea de a crea aplicaţii pentru platforme mobile dar scriind *Javascript*. Diferenţa majoră între cele doua *framework*-uri este dată de faptul că *Angular* renunţă la *scope tree*, lasând la îndemâna programatorilor doar posibilitatea de a crea componente cu scop izolat. Problema comunicării între componentele izolate ca scop se rezolvă prin aducerea unor directive numite *decoratori* (*@Input*, *@Output*).

Angular este tânăr în comparație cu *AngularJs* (2016 versus 2010) dar s-a înălțat rapid în rândul proiectelor de pe Github [5], crescând în popularitate an după an.

Majoritatea avantajelor față de *Angularjs* sunt:

- ***Typescript***. este un limbaj *statically-typed* și este superset a lui Javascript. Angular te obligă să folosesti Typescript pentru a crea o aplicație web. Unul din marele avantaje ale lui Typescript este faptul ca, fiind tradus în Javascript, se poate configura să fie compatibil cu browsere vechi (cum ar fi Internet Explorer 8);
- ***Angular-CLI*:** este un utilitar pentru linia de comandă care te ajută la dezvoltarea de aplicații în Angular. Are comenzi pentru generare de componente sau directive, testare și are inclus un utilitar numit *Webpack* pentru împachetarea diferitelor biblioteci folosite în aplicație;
- **Modularitate:** Angular se bazează mult pe modularitate deoarece ofera o logică mai bună și reduce din dimensiunea aplicației. Totodată este mentenabil și oferă extensibilitate;

3.3. *NodeJs*

NodeJs [6] este un mediu de dezvoltare *server-side* pentru *Javascript*, construit cu ajutorul motorului V8 al browser-ului Chrome.

NodeJs permite dezvoltarea de servere web și aplicații de networking folosind *Javascript* și o colecție de module de bază (sistem de fișiere; rețele - DNS, HTTP; criptografie). Pentru instalarea modulelor, se folosește *npm*, un utilitar de administrare a pachetelor Javascript, ce conține un registru de pachete a contributorilor și toată istoria versiunilor.

NodeJs procesează cererile de computație într-o buclă, procesul fiind numit *Event Loop*.

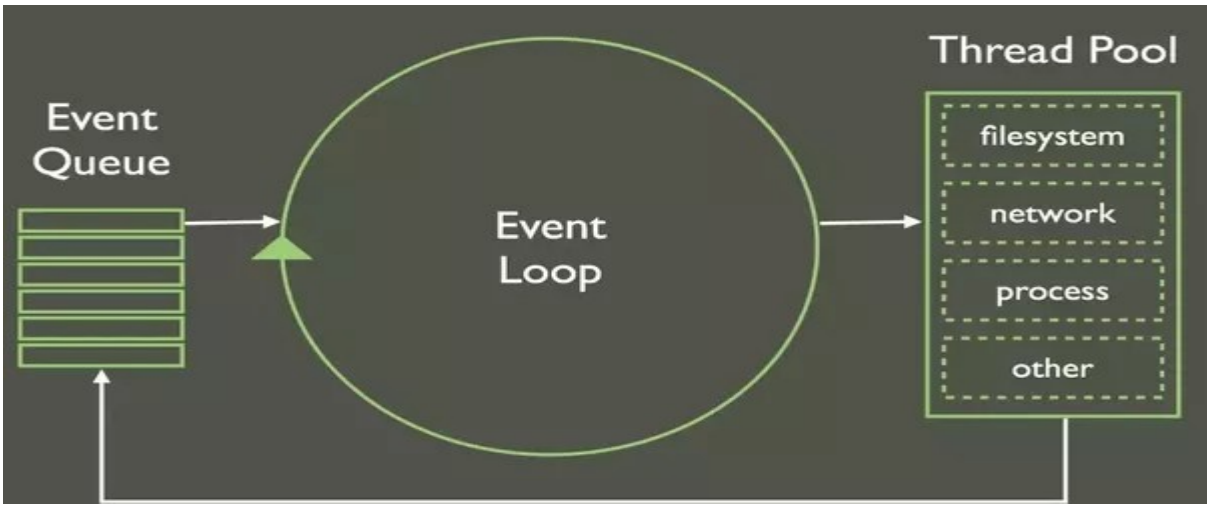

Figura 3-4 *Event Loop*

În principiu, există un singur fir de execuție, folosind operații de I/O (*Input/Output*) non-blocante și permițând suportarea a sute de conexiuni concurente. Totodată, se elimină consumul de computație al *context-switching*-ului al unui mediu în care se folosesc mai multe fire de execuție.

Acest fir de execuție principal are rolul de a administra și repartiza cererile către un *thread-pool,* o mulțime de fire de execuție care se ocupă de operații de I/O asincrone.

Un dezavantaj al acestui model de computație este existența unor evenimente costisitoare care trebuie executate de firul de execuție principal și blochează coada de evenimente.

NodeJs are câteva avantaje importante pentru alegerea dezvoltării unui server web cu această tehnologie:

- *npm*: inițializarea unui proiect se folosește prin comanda *npm install*, care citește un fișier ce descrie pachetele necesare proiectului și versiunile lor. Astfel, toate bibliotecile și datele necesare sunt aflate în directorul proiectului, fiind izolat și neexistând conflicte cu pachete globale;
- *Event Loop*: în *Javascript* toate operațiile asincrone (citirea datelor de pe *hard disk*, așteparea datelor aduse de server) se folosesc de *callbacks* pentru a informa firul de execuție principal de terminarea executării. Astfel, nu există *race conditions* sau *deadlocks* ca în cazul mediilor cu mai multe fire de execuție.
- *Javascript*: fiind un limbaj folosit atât pe *front-end* cât și pe *back-end*, va exista o comunicare mai bună între echipele din fiecare categorie, ambele având abilități asemănătoare.

3.4. *PostgreSQL*

PostgreSql [7] este un sistem de baze de date relaționale.

3.5. Biblioteci

2. Concepte teoretice și arhitectura aplicației

2. Algoritmi și structuri de date

2.2. Arhitectura aplicației

Voi scrie despre arhitectura aplicației pe front-end si back-end. Pe front-end ce componente există, rolul lor, cum interactioneaza, flow-ul aplicației. Pe back-end, routes, procesare, baza de date si ce tabele contine. (to be done)

2.3. Simulator de *debugger*

Aplicația, în decursul rulării unui cod, evidențiază linia care o execută programul, precum un *debugger*. Pentru a implementa un *debugger* real trebuie interpretat codul și ținut un tabel cu valorile variabelor și o stivă pentru apelurile funcțiilor. Nu este o cerință imposibilă luând în considerare că majoritatea *browser*-elor îl au implementat, dar pentru scopul aplicației, acela de a sincroniza pe cât posibil operațiile unei linii de cod cu visualizarea corespondentă, se poate implementa o versiune mai simplă.

La început, se va rula tot codul pe datele de intare, facând o listă cu indicii liniilor care au fost parcurse pe acele date, în ordine. Utilizatorul își poate stabili ce linii vrea să fie evidențieze când se va rula codul. Astfel, avem o listă cu toate liniile și ordinea în care vor fi parcurse, din care ne interesează doar anumiți indici.

Bibliografie

[1] "PathFinding.js". Available: http://qiao.github.io/PathFinding.js/visual/

[2] "Data Structure Visualizations", University of San Francisco.

Avalaible: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

[3] "CsAcademy Applications". Avalaible: https://csacademy.com/

[4] "Angular", Google. Avalaible: https://angular.io/

[5] "Github", Github, Inc. Avalaible: https://github.com/

[6] "NodeJs", Joyent. Avalaible: https://nodejs.org/en/

[7] "PostgreSql", PostgreSQL Global Development Group. Avalaible: https://www.postgresql.org/