**UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAŞI**

**FACULTATEA DE INFORMATICĂ**

****

LUCRARE DE LICENȚĂ

**AlgoGraph**

**propusă de**

***Schitcu Gabriel-Mihai***

**Sesiunea:** *iulie, 2019*

**Coordonator științific**

***.***

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAŞI**

**FACULTATEA DE INFORMATICĂ**

**AlgoGraph**

***Schitcu Gabriel-Mihai***

**Sesiunea:** iulie, 20189

**Coordonator științific:**

***.***

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Data \_\_\_\_\_\_\_\_\_\_\_\_ Semnătura \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemntatul(a) ………………………………………………………………………………………

domiciliul în …………………………………………………………………………………………………..

născut(ă) la data de ………………..…., identificat prin CNP ………….……………..………………..., absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de ………………………. specializarea …………………………………………………………, promoția …………………………., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_elaborată sub îndrumarea dl. / d-na \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ………………………… Semnătură student …………………………

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Titlul complet al lucrării*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de testetc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Prenume Nume*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, *Prenume Nume.*

Încheierea acestui acord este necesară din următoarele motive:

*[Se explică de ce este necesar un acord, se descriu originile resurselor utilizate în realizarea*

*produsului-program (personal, tehnologii, fonduri) și aportul adus de fiecare resursă.]*

Iași, *data*

Decan *Prenume Nume* Absolvent *Prenume Nume*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original) (semnătura în original)

Cuprins

**Introducere8**

**1 Specificatii Functionale10**

1.1 Editorul de cod11

1.2 Editorul de grafuri12

1.3 Management-ul datelor14

1.4 Flow-ul de lucru16

**2 Implementare si Testare18**

2.1 Tehnologii folosite18

2.1.1 JavaFX18

2.1.2 .Net Core18

2.1.3 Angular19

2.1.4 Alte Librarii20

2.2 Detalii tehnice20

2.2.1 Compilarea algoritmului20

2.2.2 Rularea algoritmului21

2.2.3 Colectarea comenzilor21

2.2.4 Executia comenzilor22

2.2.5 Serializare si Deserializare23

2.2.6 Generatorul de grafuri24

2.2.7 Upload si Download25

2.2.8 Management-ul bazei de date25

2.3 Testare5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

# **Introducere**

Cel mai usor mod de a intelege felul in care functioneaza anumite rutine este prin exemplu si lucru practic. Consider ca aceasta aplicatie sustine nevoia de a experimenta, oferind si o reprezentare grafica usor de urmarit, facilitand astfel buna intelegere a unor problem specifice materiei “Algoritmica Grafurilor”.

Un graf este format dintr-o multime finita de noduri si una de muchii, fiecare muchie avand rolul de a conecta o pereche de noduri. Exista doua tipuri de grafuri: grafuri orientate si neorientate. Deci, graful, ca structura de date, poate fi folosit foarte usor pentru a descrie retele si comportamente din viata reala, de la multimea strazilor dintr-un oras la diferitele relatii ce atasate unui grup de prieteni.

Modelarea grafurilor la probleme reale a dus la punerea accentului pe astfel de structuri de date. S-au dezvoltat o multitudine de algoritmi capabili sa resolve probleme specifice, de la parcurgeri si gasirea celor mai scurte cai la probleme de conectivitate si de gasire a fluxului maxim. Consider ca aceasta aplicatie aduce o reala imbunatatire la aplicatiile deja existente, ajutand atat studentii cat si alte persoane pasionate de domeniu in intelegerea mult mai usoara a unor algoritmi specifici grafurilor.

Ideea mea pentru acest proiect de licenta este dezvoltarea unei aplicatii desktop ce poate servii drept unealta educativa in materia “Algoritmica Grafurilor”. Aplicatia permite utilizatorului sa scrie sau sa incarce un algoritm specific si sa il ruleze pe un graf desenat anterior intr-o alta fereastra. Aplicatia trebuie sa ofere controale cat mai intuitive in ceea ce priveste editarea codului si a grafului. La rularea algoritmului aplicatia trebuie sa ofere utilizatorului controlul asupra vitezei la care algoritmul opereaza asupra grafului, pentru a asigura coerenta animatiilor. Totodata mi-am propus sa dezvolt si un generator de grafuri pe baza de constrangeri, pentru a ajuta la testarea rapida a algoritmilor. Fiecare graf si algoritm poate fi stocat local, insa aplicatia ofera si posibilitatea de a le incarca intr-o baza de date centralizata, de unde, dupa ce au fost acceptate de catre un administrator, vor putea fi descarcate de catre alti utilizatori.

O aplicatie asemanatoare ar fi VisuAlgo. Aceasta lucreaza pe mai multe structuri de date: liste, grafuri, vectori si altele. Fiind o aplicatie web destul de cunoscuta, VisuAlgo se foloseste de niste animatii extrem de bine puse la punct pentru a demonstra felul in care functioneaza o serie de algoritmi, am incercat sa reproduc pe cat posibil acest look&feel. Totusi exista si un mare neajuns, VisuAlgo nu permite utilizatorului sa scrie un nou algoritm si nici sa modifice unul deja existent. Acest lucru impiedica procesul de experimentare, care, din punctul meu de vedere, este crucial in buna intelegere a unor rutine.

O alta aplicatie similara este Greenfoot, menita sa ajute in invatarea programarii in limbajul Java, aceasta ofera uneltele necesare creari unor jocuri 2D. Avand un scop mult mai extins, Greenfoot m-a ajutat in a intelege cat de important este un mod inteligent de derulare a evenimentelor reprezentate grafic.

Proiectul este realizat in limbajul de programare Java. In cadrul dezvotarii aplicatiei m-am folosit de JavaFX, o platforma software dedicata creeari de aplicatii client moderne, care pot fi rulate pe o gama larga de sisteme de operare. Am mai utilizat RichTextFX, o librarie specifica JavaFX, pentru a avea acces la evidentierea sintaxei, numerotarea liniilor si alte caracterisitici necesare unui editor de cod modern. Majoritatea stilizarilor din aplicatie au fost realizate programatic, prin metodele JavaFX, insa, pentru o mai buna organizare am folosit si mici fragmente de stilizare css.

Pentru compilarea si rularea codului la runtime m-am folosit de clasa JavaCompiller oferita de catre system si de Reflection, un API Java folosit pentru a modifica, examina si rula metode ale unor obiecte java la runtime. Prin Reflection am avut posibilitatea de a instantia clasa fisierului rezultat in urma compilarii codului scris de catre utilizator. Formatarea codului respectiv a fost posibila folosind API-ul Google Java Format, care formateaza surse java dupa standardele cerute de catre Google.

Baza de date centralizata, creata in Microsoft SQL Server, si Data Storage-ul pentru blob-uri de date au fost hostate in Cloud-ul Microsoft Azure. Pentru management-ul bazei de date m-am folosit de un REST Api implementat in framework-ul .Net Core, expunand si o interfata web destinata administratorilor realizata in Angular.

# **Specificatii Functionale**

Scopul aplicatiei este dezvoltarea unei aplicatii desktop capabila sa ofere utilizatorilor o platforma prietenoasa in ceea ce priveste scrierea si reprezentarea pasilor unor algoritmi specifici grafurilor. Odata cu rularea unui algoritm aplicatia trebuie sa ofere utilizatorului posibilitatea de a controla dupa bunul plac modul si viteza cu care respectivul algoritm manipuleaza graful. Utilizatorul poate alege sa suspende flow-ul algoritmului sau sa avanseze manual sau chiar automat prin pasii acestuia.

O alta functionalitate majora este generatorul de grafuri pe baza de constrangeri. Odata pornit, acesta interogheaza utilizatorul cu privire la constrangerile dorite, ca mai apoi sa incerce sa genereze un graf ce se incadreaza in limitele impuse de catre utilizator. Aceste constrangeri ar putea viza numarul de noduri sau muchii, conectivitatea grafului, valorile etichetelor componentelor si altele. Aplicatia ofera si posibilitatea de a face sharing de grafuri si algoritmi, utilizatorul fiind capabil sa incarce resurse care, dupa ce au fost acceptate de catre un administrator, vor putea fi descarcate si folosite de catre toti utilizatorii aplicatiei.

Am dedus ca pentru a duce la sfarsit tot ceea ce mi-am propus voi avea nevoie sa dezvolt 4 mari module: un editor de cod, un editor de grafuri, un sistem solid de management a resurselor utilizatorului si o infrastructura inteligenta care face legatura intre cod si graf, rezultand o serie de animatii coerente.

## **Editorul de cod**

Editorul de cod este o fereastra ce permite utilizatorului sa isi scrie propriul program ce va manipula un graf. Am ales limbajul de programare Java, ca limbaj folosit de utilizator pentru contructia algoritmilor, fiind modern, cunoscut si cu o sintaxa prietenoasa. Permite toate operatiunile necesare unui editor de text: copiere, taiere, lipire, selectare etcetera. Fiind dezvoltat cu ajutorul RichTextFX, acesta profita de anumite caracteristici importante ca evidentierea sintaxei si numerotarea liniilor. Acest editor prezinta urmatoarele butoane si functii:

**Save**: permite salvarea progresului intr-un fisier ce poate fi ulterior deschis.

**Compile**: compileaza codul sursa scris de catre utilizator, este necesara apasarea lui inainte de rulare. In cazul in care exista erori de compilare editorul de cod va evidentia acest lucru printr-un semn specific la linia cu pricina.

**Format**: preia codul sursa scris de catre utilizator si il formateaza comform standardelor impuse de Google.

**Run**: se foloseste de fisierul produs prin compilarea codului si, avand acces la graful desenat in cealalta fereastra, incepe procesul de reprezentare grafica a algoritmului.

## **Editorul de grafuri**

Este o functionalitate extrem de importanta in aplicatie. Pe langa necesitatea evidenta de a permite utilizatorului sa creeze grafuri dupa bunul plac, editorul de grafuri reprezinta totodata si locul in care se desfasoara toate animatiile si manipularile survenite in urma rularii unui algoritm.

Exista mai multe librarii care se ocupa cu reprezentarea grafica a grafurilor intr-o aplicatie java. Una dintre acestea este JGraphX care, avand la baza structura de date definite de libraria JGraphT, poate afisa cu usurinta grafurile necesare. Totusi, acestea nu au suport JavaFX, ci doar Swing. Desi exista posibilitatea importului de componente Swing intr-o aplicatie JavaFX, am decis ca din pricina acestui inconvenient dar si a altor motive precum aspectul nesatisfacator si lipsa controlului total asupra functionalitatii grafurilor, imi voi dezvolta propria modalitate de a reprezenta grafuri in JavaFX.

In ceea ce priveste infatisarea grafului am optat pentru o tema usor de inteles, punand pe primul loc coerenta si vizibilitatea datelor. Ca paleta initiala de culori am ales alb si negru, sperand sa scot in evidenta animatiile si culorile alese de catre utilizator.

Nodurile sunt reprezentate vizual de cercuri albe cu un contur negru. La dorinta utilizatorului, nodurile pot contine o etucheta cu un nume sau o valoare numerica. Centrata in interiorul nodului, aceasta eticheta este de o culoare cenusie transparenta, pentru a nu obstruction prea mult reprezentarea nodului.

Muchiile, pe de alta parte, sunt linii negre ce conecteaza doua noduri. La fel ca nodurile, muchiile pot sau nu avea etichete cu valori, care, de aceasta data, vor fi asezate la mijlocul liniei. In cazul in care avem de-a face cu un graf orientat, muchiile vor arata “cursul” sursa-destinatie printr-o sageata prezenta la unul dintre capete. In cazul in care muchia respective este una dubla, reprezentarea grafica a acesteia va infatisa sageti la ambele capete.

Graful rezultat, reprezentat in editor, este in totalitate responsive. Utilizatorul poate modifica pozitia oricaror componente prin drag and drop, iar starea componentelor afectate se va schimba in mod corespunzator. Editorul de grafuri prezinta urmatoarele butoane si functii:

**Save**: similar editorului de cod, permite salvarea locala a progresului realizat in graful curent.

**Add Edges**: schimba modul de lucru asupra grafului pentru a permite utilizatorului sa adauge muchii. O muchie este adaugata atunci cand doua noduri sunt selectate consecutive. Primul nod reprezinta sursa, iar cel de-al doilea destinatia.

**Add Nodes**: Permite adaugarea de noduri, schimband modul de lucru asupra grafului. Un nod este adaugat la apasarea pe o zona libera din editor.

**Delete Components**: Permite stergerea de muchii si noduri odata cu selectarea acestora. O apasare dubla a acestui buton va conduce la stergerea intregului graf.

**Edit**: Fiecare componenta a grafului are o eticheta cu valori care este ascunsa atunci cand component nu are valoare. Acest buton faciliteaza aparitia etichetelor pentru a putea fi editate.

**Switch**: Schimba arhitectura grafului din orientat in neorientat si invers. Este atat o schimbare de reprezentare grafica cat si una pe partea de “back-end”.

**Generate**: Porneste generatorul de grafuri pe baza de constrangeri. Voi discuta modul in care functioneaza mai jos.

## **Management-ul datelor**

Atat algoritmii scrisi, cat si grafurile desenate pot fi salvate local si incarcate la dorinta utilizatorului. Acestea vor fi stocate sub forma de fisiere intr-un folder vecin al proiectului, fiecare fisier avand un nume unic selectat de utilizator. Algoritmii sunt salvati ca fisiere .java, facand astfel mai usoara compilarea si utilizarea lor. Avand in vedere simplitatea resursei folosite de editorul de cod (algoritumul fiind un simplu string), aceasta a fost cea mai intuitiva varianta.

Pe de alta parte, in cazul grafurilor nu am avut o solutie asa de usoara. Am inceput prin a le serializa binar si stoca in fisiere. Aceasta varianta a fost foarte usor de implementat, serializarea obiectelor in java este usoara si bine documentata. Totusi, exista cateva neajunsuri. In cazul aplicatiei mele, clasa grafului este una in continua schimbare, in sensul in care comportamentul acesteia este imbogatit regulat. Acest lucru nu este suportat de catre serializarea binara, intrucat aceasta descrie un model anterior al clasei, asadar la fiecare schimbare a clasei Graph, toate obiectele serializate anterior vor fi pierdute.

Cautand o solutie adecvata am descoperit formatul de fisier GraphML. Acesta este un format intuitiv si unor de folosit pentru grafuri. Are suport pentru grafuri orientate si neorientate, si este usor de extins pentru a include diferite atribute nodurilor si muchiilor. Am cautat o biblioteca capabila sa parseze un graf si sa imi ofere serializarea acesteia GraphML, insa nu am gasit unul potrivit, fiecare fiind specializat pe grafuri descrise de librarii ca JGraphX sau, in cazul celor care puteau lucra cu un graf custom, adaugau prea multe atribute nefolosite de modelul meu de graf, rezultand o serializare greu de urmarit.

Asadar, mi-am facut propriul parser de grafuri, capabil sa transforme modelul meu de graf intr-un fisier GraphML si viceversa. GraphML este un format bazat pe XML, deci nu are o sintaxa stricta, urmand setul de reguli propus de documentatie am reusit sa fac un parser satisfacator. Cu ajutorul acestuia am rezolvat problema schimbarii modelului de graf, atata timp cat nu modific definitia grafului in sine (o multime de noduri si o multime de muchii ce conecteaza perechi de noduri), pot adauga si modifica functionalitati, serializarea GraphML v-a descrie corect modelul de graf. Odata cu acest avantaj mai apare unul. Standardul GraphML este destul de folosit si in alte aplicatii de constructie de grafuri, ceea ce inseamna ca este foarte usor sa exportam un graf din NumeAplicatie si sa il folosim in respectivele aplicatii si invers. Un user poate prelua un graf interesant si il poate incarca in NumeAplicatie, atata timp cat serializarea acestuia respecta standardul GraphML, lucru imposibil de realizat folosind serializarea binara.

## **Flow-ul de lucru**

Ce se intampla cu adevarat la apasarea butonului “Run”?

Pentru inceput se verifica existenta fisierului produs in urma compilarii codului scris de catre utilizator. Apoi, folosindu-ne de Reflection API cream o instanta a clasei descrise de acel fisier. Un lucru foarte important de mentionat este ca obiectul rezultat trebuie sa fie o instanta a clasei Program, astfel incat putem fi siguri de implementarea unor metode cruciale in rularea algoritmului, si anume, a metodei “run”. Daca aceste conditii sunt indeplinite vom avansa la urmatoarea etapa.

Aplicatia isi propune reprezentarea manipularii grafului intr-un mod controlat de catre utilizator. Asta presupune derularea manuala sau, setand un interval, automata a comenzilor asupra grafului. Deci nu putem permite algoritmului sa lucreze direct pe graful din editor, intrucat asta ar insemna ca modificarile asupra grafului vor avea loc odata cu rularea codului, mult prea rapid pentru a putea fi urmarite de catre utilizator. Aceasta problema poate fi rezolvata in doua moduri:

* Putem “coopera” cu utilizatorul. Instruind utilizatorul sa foloaseasca apeluri de sleep ale sistemului pentru a creea iluzia de parcurgere secventiala a comenzilor asupra grafului. Totusi, acest approach nupermite derularea manuala si nici pauza, insa cel mai mare neajuns este nevoia de a restrictiona utilizatorul din a scrie cod curat si usor de urmarit.
* Putem reduce graful reprezentat “grafic” la o structura de date “dummy” mai simpla. Astfel, atat graful cat si toate componentele sale sunt reduse la obiecte ce implementeaza comportamentul unui graf, si care ofera posibilitatea adaugarii de functionalitate. Folosindu-ne de asta am putea modifica obiectele “dummy” in mici istoricuri, capabile sa memorize toate evenimentele produse de catre utilizator. Astfel, algoritmul utilizatorului este rulat pe un graf dummy ce joaca rolul de istoric, iar la final putem culege toate evenimentele, ca mai apoi sa fie rulate pe graful autentic, dupa bunul plac.

Evident, am ales varianta a doua. Astfel, preluam graful din editor si ne folosim de el pentru a creea un graf dummy. In continuare algoritmul utilizatorului este rulat prin Reflection API, oferindu-I graful dummy ca parametru. Rezultatul rularii este o lista de comenzi, care, puse in ordine descriu perfect evenimentele ce trebuie parcurse pentru a respecta algoritmul utilizatorului. Aceasta lista de comenzi este pasata mai departe unei clase CommandsRunner care, folosindu-se de input-ul utilizatorului, ruleaza comenzile in modul dorit. Vom discuta amanuntit aceasta functionalitate mai jos.

# **Testare si Implementare**

## **2.1 Tehnologii Folosite**

## **2.1.1 JavaFX**

JavaFX este un set de pachete grafice care permit unui developer sa creeze, testeze si publice aplicatii desktop RCA (Rich Client Application) ce pot functiona pe diverse sisteme de operare. Fiind un API Java, o aplicatie JavaFX permite programatorului sa utilizeze oricare alte librarii si API-uri Java native, fiind capabila sa acceseze capabilitatile sistemului sau sa se conecteze la alte aplicatii sau servere.

Pentru a separa design-ul de restul aplicatiei, stilizarile pot fi facute fisiere css care pot fi applicate ulterior. Constructia scenei este facuta fie programmatic fie prin limbajul de scriptare FXML sau chiar prin drag&drop cu ajutorul JavaFX Scene Builder. JavaFX detine o multitudine de controale built-in (butoane, etichete, text input …) care pot fi folosite pentru o mai usoara dezvoltare a scenei. Fiind o solutie menita sa inlocuiasca Swing (desi inca nu a facut-o), JavaFX permite interoperabilitatea si conversia acestui tip de aplicatie, dar si utilizarea componentelor Swing in cadrul proiectului.

In cadrul aplicatiei m-am folosit in principal de mediul de dezvoltare JavaFX pentru cea mai mare parte a aplicatiei, cu preacadere in constructia interfetei grafice, folosind o multitudine de controale specific JavaFX dar si librarii dezvoltate ulterior, menite sa imbogateasca capabilitatile utilizatorului, precum RichTextFX. Pentru desenari mai complexe ale grafului m-am folosit de Canvas API, un feature built-in JavaFX care permite desenarea a diferite forme geometrice pe o arie din scena aplicatiei. Desi aveam si alte solutii mai usoare am ales sa construiesc scena principala a aplicatiei in mod programmatic, avand aceeasi solutie si in cadrul stilizarii, cu mici exceptii.

## **2.1.2 .Net Core**

.Net Core este un framework open-source capabil sa creeze aplicatii cross-platform conectate la internet si bazate pe cloud. Printre beneficiile .Net coe se Numara: o arhitectura usor testabila, abilitatea de a rula atat pe Windows, macOS si Linux, posibilitatea de integrare cu framework-uri pe partea de client, dependency injectrion este un feature built-in, posibilitatea de a fi hostat pe diferite servere, un manager de pachete modern si unelte ce simplifica procesul de web development.

Folosind .Net Core am realizat un API REST, o aplicatie web menita sa expuna informatii despre un set de resurse, oferind posibilitatea creari, modificarii si interogarii respectivelor resurse. Pentru comunicarea cu baza de date am folosit Entity Framework Core, un framework ce permite maparea tabelelor la entitati .Net, usurand foarte mult dezvoltarea aplicatiei.

## **2.1.3 Angular**

Fiind un framework folosit in crearea de aplicatii client, Angular este un set de librarii care, utilizand HTML, css si TypeScript, o variatie tipizata a limbajului JavaScript, faciliteaza si structureaza procesul de dezvoltare web. Pentru a usura in asezarea fundatiilor unei aplicatii web, Angular ofera, printer altele, trei unelte:

**Modulul** descrie un context de compilare si un set de componente ce descriu un workflow, o functionalitate coerente. Foarte folositor in structurarea proiectului, modulele permit exportul si importul de functionalitate cu alte module, o rutare moderna si suport pentru lazy-loading, o practica ce permite incarcarea codului strict necesar, minimizand costurile.

**Componentele** sunt o asociere intre logica de business, scrisa intr-o clasa TypeScript, si view reprezentat printr-un fisier HTML. Capabile sa foloseasca alte componente, acestea permit constructia unui view complex si extensibil, facilitand reutilizarea codului.

**Serviciile** sunt simple clase TypeScript care nu sunt asociate unui view. Acestea sunt importate in diferite componente cu ajutorul Dependency Injection si pot descrie o gama variata de functionalitate, de la constituirea unui element de legatura intre componente la clase care se ocupa cu comunicarea cu un API.

## **2.1.4 Alte librarii**

In cadrul proiectului am folosit diferite librarii menite sa imi usureze munca. RichTextFX este una dintre acestea, o librarie destinata aplicatiilor JavaFX, aceasta ofera niste unelte avansate in dezvoltarea unui editor de text modern. Cu ajutorul acesteia cat si a unor tutoriale explicative gasite pe pagina de github am reusit sa creez editorul de cod, o componenta foarte importanta in aplicatie. RichTextFX ofera o interfata usor de folosit in ceea ce priveste stilizarea textului, lucru greu de facut cu primitive JavaFX. Cu ajutorul librariei am reusit sa implementez evidentierea sintaxei, numerotarea liniilor si evidentierea liniilor cu probleme. Pentru formatarea codului am utilizat Google Java Format, un API ce formateaza cod Java dupa standardele Google. Deoarece API-ul ofera posibilitatea formatarii unui string, aceasta operatiune a fost simplu de executat, inlocuind codul din editor cu textul rezultat din formatarea codului respectiv.

Jersey este un framework open-source utilizat pentru dezvoltarea serviciilor web REST. Cu toate acestea eu nu m-am folosit de aceasta latura, alegand sa implementez API-ul REST in .Net Core, insa am folosit client-ul HTTP oferit de Jersey 2, JAX-RS. Acest client poate fi folosit pentru a face diferite request-uri HTTP catre un WebTarget. In aplicatie acesta este folosit pentru a trimite si primii date cu privire la grafuri si algoritmi de la un API.

## **2.2 Detalii Tehnice**

## **2.2.1 Compilarea algoritmului**

Algoritmul utilizatorului este stocat local intr-un fisier .java, facilitand usurinta compilarii si a editarii. Am implementat clasa ProgramCompiller care, folosindu-se de o instanta JavaCompiller oferita de catre Environment, poate compila fisiere java a caror cale a fost primita anterior. Aceasta creeaza un CompilationTask care, odata rulat, genereaza fisierul .class corespunzator si stocheaza anumite rezultate intr-un DiagnosticsContainer. Daca este cazul, erorile la compilare vor fi stocate in acel container care este returnat de catre metoda implementata de mine.

Odata ce task-ul de compilare este complet verific diagnosticul si, in cazul in care exista erori de compilare, aflu numarul liniei cu pricina. Folosindu-ma de una din functionalitatile RichTextFx creez un mapping observabil care dicteaza ca orice linie a carei numar este egal cu numarul liniei cu eroarea gasita anterior v-a prezenta un mic icon ce o evidentiaza. Un mapping asemanator folosesc si pentru numerotarea liniilor.

## **Rularea algoritmului**

Dupa compilarea cu success a programului scris de catre utilizator, un fisier .class este generat la o cale cunoscuta de catre aplicatie. Aceasta clasa este preluata de catre aplicatie si instantiate folosindu-ne de Reflection API. Algoritmul scris de catre utilizator este fortat sa implementeze clasa abstracta Program, deci in continuare este verificata aceasta conditie. Avand confirmarea necesara stim ca instant clasei detine o metoda run(graph) pe care o putem instantia si rula dupa bunul plac.

## **Colectarea comenzilor**

O comanda este o actiune de manipulare a grafului survenita in urma unor metode apelate de catre algoritmul utilizatorului. O astfel de comanda poate definii accesarea unor noduri, stergerea unei muchii, schimbarea culorii unei componente, printarea unui mesaj la consola si multe altele. Fiecare comanda extinde clasa abstracta Command. Acest lucru obliga clasa sa detina o variabila commandOrder care defineste ordinea comenzii in intreaga colectie de comenzi survenite in urma rularii algoritmului. La executia comenzii aceasta primeste graful pe care trebuie sa opereze si un numar ce semnifica durata animatiei. Aceasta durata este seta de utilizator printr-un slider, in functie de viteza dorita.

Intrucat nu vrem sa rulam algortimul direct pe graful din editor (viteza manipularii grafului ar fi mult prea mare pentru utilizator), am dezvoltat o functionalitate care mapeaza graful si fiecare componenta a sa la un obiect separat. Initial acest dummy reflecta exact structura de date a grafului din editor, dar, odata trimis metodei scrise de catre utilizator, devine complet independent. Fiecare componenta a grafului trimis catre utilizator, cat si graful in sine, contine o colectie de comenzi care actioneaza ca un istoric al actiunilor luate de catre algoritm asupra structurii de date. Asadar, orice actiune asupra unei componente din graf v-a fi tradusa ca fiind o comanda. Drept exemplu, apelul setColor asupra unui nod va conduce la adaugarea unei comenzi de tipul setNodeColorCommand in setul de comenzi al nodului respectiv.

Crearea unei comenzi presupune si generarea unui commandOrder, adica a unui numar ce indica ordinea comenzii in intregul ansamblu de actiuni. Acest commandOrder este obtinut dintr-un camp static al grafului, incrementat la crearea unei noi comenzi. La finalul rularii algoritmului aplicatia trebuie sa centralizeze toate comenzile survenite, interogand atat graful cat si fiecare nod si muchie a sa. Astfel obtinem o lista de comenzi care, odata sortata crescator dupa campul commandOrder, descrie perfect toate tranzitiile prin care a trecut graful de la inceputul pana la sfarsitul algoritmului.

## **2.2.4 Executia comenzilor**

Dupa rularea algoritmului scris de catre utilizator obtinem un set ordonat de comenzi ce trebuie executate asupra grafului din editor. Fiecare comanda implementeaza metoda run care descrie modul in care aceasta manipuleaza graful, in acelasi timp aceasta metoda este responsabila pentru definirea animatiei corespunzatoare comenzii. Deoarece intre executia a doua comenzi v-a exista un delay de lungime variabila nu putem parcurge comenzile pe firul de executie principal, apelul unor metode de tipul sleep ar suspenda procesele JavaFX. Asadar CommandsRunner, clasa care se ocupa cu preluarea, management-ul si executia comenzilor asupra grafului, extinde clasa Thread, lucrand in paralel cu thread-ul principal JavaFX.

Aplicatia pune la dispozitie si un panou de control destinat utilizatorului. Initial acesta este dezactivat, insa dupa executia unui algoritm panoul devine utilizabil. Pe langa niste view-uri informative legate de numarul de comenzi si timpul de executie al algoritmului, panoul prezinta urmatoarele controale: un slider, reprezentand durata delay-ului intre executia a doua comenzi, si 4 butoane:

**Pause**: suspenda executia comenzilor.

**Play**: porneste executia automata a comenzilor, delay-ul fiind cel setat in slider.

**Manual Step**: inainteaza cu o comanda apoi se schimba in Pause.

**Exit**: opreste executia comenzilor.

Pentru a fi mereu la curent cu alegerile utilizatorului, clasa commandsRunner detine o instanta de panou de control pe care o consulta ori de cate ori trebuie sa faca o decizie. Slider-ul poate fi modificat oricand, intrucat exista un binding, o proprietate ce permite sincronizarea a doua variabile, intre acesta si valoarea folosita ca delay. Lucrul care ne permite o buna tranzitie intre modurile de executie a comenzilor (Automatic, Manual, Pause) este un obiect de tipul lock. Acesta suspenda si porneste firul de executie in functie de alegerea utilizatorului.

## **Serializare si Deserializare**

Pentru modul de serializare al grafurilor am ales standardul GraphML, bazat pe XML acest format este prietenos si folosit de multe alte editoare de grafuri. Intrucat nu am gasit un convertor de grafuri custom in GraphML, am ales sa imi creez propria solutie capabila sa importe si sa exporte grafuri in fisiere graphml. Acest lucru este implementat prin clasa GraphIO, punand la dispozitie metodele importGraph si exportGraph.

Fiind un fisier XML, constructia sa este destul de usor de realizat folosindu-ma de clasa Document, care faciliteaza usoara adaugare de noduri si valori specifice acestui tip de fisier. Standardul GraphML suporta trei tipuri de entitati: grafuri, noduri si muchii. Insa, inainte de a adauga aceste entitati trebuie sa adaugam o serie de noduri atribut, fiecare avand un id si o entitate tinta. In cazul meu am 5 astfel de atribute: valoarea unui nod, valoarea unei muchii, pozitia unui nod (x si y) si culoarea unui nod. Aceste atribute vor fi gasite drept noduri XML in cadrul entitatilor pe care le descriu. Fiecare entitate, fie un graf, o muchie sau un nod, are nevoie de un id numeric cu ajutorul caruia poate fi referita in diferite relatii. Drept exemplu, o muchie trebuie sa prezinte un atribut source si unul target, valorile acestor attribute fiind id-urile nodurilor ce indeplinesc acel rol.

## **Generatorul de grafuri**

Acesta are doua mari componente: interfata grafica si generatorul in sine. Interfata grafica este o noua scena care, odata activata, apare ca un popup pe ecranul utilizatorului. Aceasta prezinta diferite variatii de constrangerii sub forma de checkmark-uri, la selectarea unei constrangeri vor aparea in view si input-urile specifice ei. Avem trei tipuri mari de constrangeri:

**Boolean Constraint**: denota o constrangere de tipul adevarat sau fals, care nu necesita introducerea unor limite. Exemplu: caracteristica conexitatii grafului rezultat.

**Single Value Constraint**: este o constrangere ce se foloseste de o singura limita setata de utilizator: Exemplu: lungimea celui mai mare circuit din graf.

**Double Value Constraint**: reprezinta tipul de constrangere cu o limita inferioara si una superioara. Exemplu: numarul de noduri din graful generat.

Fiecare constrangere trebuie sa implementeze clasa abstracta Constraint, astfel fiind fortate sa implementeze metoda check(graph) care testeaza nivelul de completitudine al constrangerii pentru graful dat. Feedback-ul oferit de aceasta metoda este de trei feluri: Complete, Incomplete si Revert. Evident, Complete denota indeplinirea constrangerii iar Incomplete ne spune ca, desi constrangerea nu este indeplinita in mod current, exista inca posiblitatea ca aceasta sa fie respectata in viitor. Pe de alta parte, Revert inseamna ca limitele constrangerii au fost depasite si ca nu mai exista posibilitatea indeplinirii acesteia in viitor, astfel este necesar rollback-ul ultimului pas din generare.

Dupa selectarea constrangerilor si a limitelor dorite, utilizatorul poate incepe generarea prin apasarea butonului “Generate”. Odata cu aceasta actiune aplicatia interogheaza interfata grafica si isi creaza o lista de constrangeri pe care o paseaza mai departe generatorului de grafuri. Procesul de generare de grafuri este urmatorul. Aplicatia genereaza aleator un pas de manipulare a grafului, executandu-l asupra grafului. Intreaga lista de constrangeri este interogata pentru a decide urmatoarea actiune. Daca toate constrangerile sunt complete atunci generarea a luat sfarsit, insa daca una din constrangeri returneaza feedback-ul Revert, ultimul pas este anulat si generarea continua. Daca vreo constrangere returneaza Incomplete, generarea va continua, feedback-ul primit insemnand ca inca exista posibilitatea indeplinirii tuturor constrangerilor.

Fiecare pas din procesul de generare al grafurilor trebuie sa implementeze interfata IStep, asigurandu-ne astfel ca fiecare pas ofera posibilitatea de executie si rollback. Generarea de grafuri se incheie atunci cand toate constrangerile sunt indeplinite sau cand un anumit numar de pasi au fost executati fara a duce la o solutie, indicand faptul ca limitele utilizatorului nu pot fi respectate.

## **2.2.7 Upload si Download**

Aplicatia ofera utilizatorului posibilitatea de a incarca algoritmi si grafuri intr-o baza de date centralizata astfel incat, prin intermediul unui panou de browsing, alti utilizatori le pot descarca si folosii. Baza de date este administrate de catre un API Rest care expune cateva rute folosite de catre aplicatie pentru a posta si lua algoritmi si grafuri. Pentru a face cereri la API din aplicatie m-am folosit de framework-ul Jersey, utilizand un client JAX-RS.

Ambele editoare prezinta cate un buton de upload care poate fi actionat doar cand utilizatorul lucreaza cu un algoritm sau graf salvat. La apasarea lui aplicatia mapeaza resursa respective la un obiect ApiEntity si se foloseste de un client JAX-RS pentru a face un request POST catre API. Pe de alta parte, pentru a descarca astfel de resurse, utilizatorul trebuie sa selecteze item-ul “Browse” al meniului, care odata actionat preia toate resursele necesare de la API si le afiseaza intr-un nou panou, de unde pot fi descarcate cu usurinta.

### **2.2.8 Management-ul bazei de date**

Baza de date, realizata in Microsoft SQL Server are o interfata spre exterior realizata printr-un API REST in .Net Core. Aceasta contine doua tabele, unul pentru grafuri si unul pentru programe. Pentru a stoca corpul unei astfel de resurse m-am folosit de un Data Storage pentru blob-uri hostat in Microsoft Azure, astfel fiecare entitate din tabele contine cate un id in plus care face legatura intre ea si blob-ul asociat ei din Data Storage.

API-ul detine doua controllere, cate unul pentru fiecare resursa. Acestea expun diferite rute ce conduc catre operatii specifice unui API REST, precum adaugarea, stergerea, interogarea si modificarea unei entitati. Cu exceptia rutelor de adaugare si interogare de resurse active, toate rutele necesita autentificare pentru a putea fi accesate. Aceasta autentificare este de tipul Basic Authentication, o solutie simpla ce permite logarea unui utilizator administrator.

Comunicarea cu baza de date este realizata prin Entity Framework Core. Acesta permite maparea entitatilor stocate in tabele la clase specifice .Net, astfel faciliteaza paradigmele programarii orientate obiect, facand posibila interogarea si modificarea bazei de date intr-un mod programatic.

Pentru a oferii o interfata grafica atractiva, AlgoGraph pune la dispozitie o aplicatie web dezvoltata in Angular ce permite management-ul resurselor.