**ROMÂNIA**

**MINISTERUL APĂRĂRII NAŢIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare și sisteme informatice pentru apărare și securitate națională**

**PLATFORMĂ EDUCATIVĂ PENTRU TESTARE CU ÎNTREBĂRI GENERATE PROCEDURAL**

ABSOLVENT:

**Std. Sg. Maj. Porfirie-Denissa PILIPĂUȚANU**

CONDUCĂTOR ŞTIINŢIFIC:

**Ș.L. dr. ing. Cristian CHILIPIREA**

Conține \_\_\_\_\_\_\_\_\_\_ file

Inventariat sub nr. \_\_\_\_\_

Poziția din indicator: \_\_\_\_\_

Termen de păstrare: \_\_\_\_\_\_

**BUCUREŞTI**

**2022**

# Abstract

With an increasing number of students enrolled in programming classes, it is a well-known fact that the teachers regularly face the challenge of producing new knowledge assessment content due to the publication of existing materials and open access to them. Thus, a need to create a large set of exercises in a short time arises.

Therefore, this paper aims to present an automated evaluation system that ensures the variety of content through the procedural generation of questions and their answers. As a consequence of the fact that the project was designed in order to innovate the evaluation method within the Data Structures and Algorithms course, it is focused on the generation of programming exercises.

There are a variety of types of exercises that can be used to assess a student's programming skills. The paper addressed the exercises that assess the knowledge and application of algorithms, as well as the semantic exercises that consist in understanding the flow of a program.

# Rezumat

Formarea unui număr mare de specialiști devine o provocare pentru resursa umană limitată din mediul universitar. Profesorii se confruntă în mod regulat cu sarcina de a crea itemi pentru testare din cauza publicării materialelor existente și accesului deschis la acestea. Astfel, apare nevoia de a crea într-un timp scurt un set amplu de exerciții.

Prin urmare, această lucrare își propune să prezinte un sistem de evaluare automatizat ce asigură varietatea de conținut prin intermediul generării procedurale a întrebărilor și a răspunsurilor aferente acestora. Ca o consecință a faptului că proiectul a fost conceput cu scopul de a inova modalitatea de evaluare din cadrul cursului de Structuri de Date și Algoritmi, acesta este focusat asupra generării exercițiilor de programare.

Există o multitudine de tipuri de exerciții care pot fi folosite pentru a evalua cunoștințele de programare ale unui student. În cadrul lucrării au fost abordate exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi, precum și exercițiile semantice ce constau în înțelegerea fluxului unui program.

# Cuprins

[1. Introducere 1](#_Toc107332675)

[2. Literatură relevantă recentă 5](#_Toc107332676)

[2.1. Metode generative pentru întrebările cu răspuns liber 5](#_Toc107332677)

[2.2. Soluții ce utilizează șabloane parametrizate 7](#_Toc107332678)

[2.3. Soluții ce utilizează noduri de instrucțiuni 9](#_Toc107332679)

[2.4. Soluții ce utilizează gramatici (biblioteca Tracery.js) 11](#_Toc107332680)

[2.5. Soluții comerciale similare (ASC, Brio) 13](#_Toc107332681)

[3. Noțiuni generale de funcționare a unui compilator 15](#_Toc107332682)

[3.1. Etapele procesului de compilare 15](#_Toc107332683)

[3.2. Analiza sintactică 17](#_Toc107332684)

[3.3. Generarea de cod intermediar 20](#_Toc107332685)

[3.4. Tabela de simboluri 23](#_Toc107332686)

[4. Detalii privind implementarea practică 24](#_Toc107332687)

[4.1. Arhitectura sistemului 25](#_Toc107332688)

[4.2. Tehnologii și API-uri folosite 26](#_Toc107332689)

[4.3. Implementarea părții de frontend 27](#_Toc107332690)

[4.4. Implementarea părții de backend 29](#_Toc107332691)

[Crearea API-ului 29](#_Toc107332692)

[Implementarea autentificării JWT 31](#_Toc107332693)

[Interacțiunea cu baza de date 32](#_Toc107332694)

[Implementarea serviciilor 35](#_Toc107332695)

[4.5. Modul de funcționare al generatorului de întrebări 37](#_Toc107332696)

[4.6. Descrierea testelor 46](#_Toc107332697)

[5. Rezultate experimentale 47](#_Toc107332698)

[6. Concluzii 48](#_Toc107332699)

[Bibliografie 49](#_Toc107332700)

# Listă figuri

[Figură 2‑1 - Secvențe de cod generate de James Tiam-Lee și Kaoru Sumi 11](#_Toc107332701)

[Figură 2‑2 - Secvențe de cod generate în cadrul acestui proiect 11](#_Toc107332702)

[Figură 2‑3 - Interfața de creare a unor itemi ce vor fi generați automat în cadrul platformei ASC 13](#_Toc107332703)

[Figură 2‑4 - Itemii generați automat pentru un test în cadrul platformei Brio 14](#_Toc107332704)

[Figură 3‑1 - Structura unei instrucțiuni if-else 17](#_Toc107332705)

[Figură 3‑2 - Regula de producție pentru o instrucțiune if-else 17](#_Toc107332706)

[Figură 3‑3 - Gramatica unui limbaj de programare primitiv ce generează un set limitat de instrucțiuni 18](#_Toc107332707)

[Figură 3‑4 - Arborele de sintaxă al neterminalului A 19](#_Toc107332708)

[Figură 3‑5 - Arborele de sintaxă al unei instrucțiuni if-else 19](#_Toc107332709)

[Figură 3‑6 - Valorile atributelor nodurilor arborelui de sintaxă unei expresii 21](#_Toc107332710)

[Figură 3‑7 - Pseudocodul unei traversări în adâncime a unui arbore de sintaxă și al evaluării atributelor nodurilor 22](#_Toc107332711)

[Figură 4‑1 - Diagrama cazurilor de utilizare a platformei educative 24](#_Toc107332712)

[Figură 4‑2 - Arhitectura platformei educative de testare 25](#_Toc107332713)

[Figură 4‑3 - Fluxul acțiunilor în modelul arhitectural MVC 26](#_Toc107332714)

[Figură 4‑4 - Componentele create pentru interfața platformei educative 28](#_Toc107332715)

[Figură 4‑5 - Procesul de autentificare și autorizare 32](#_Toc107332716)

[Figură 4‑6 - Diagrama bazei de date 33](#_Toc107332717)

[Figură 4‑7 - Fluxul de transmitere al informației 34](#_Toc107332718)

[Figură 4‑8 - Poziționarea serviciilor în fluxul de lucru al aplicației 35](#_Toc107332719)

[Figură 4‑9 - Schema generală a generatorului pentru exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi 37](#_Toc107332720)

[Figură 4‑10 - Schema generală a generatorului pentru exercițiile semantice 38](#_Toc107332721)

[Figură 4‑11 - Diagrama de activitate a generatorului 38](#_Toc107332722)

[Figură 4‑12 - Diagrama de activitate a pasului de generare a funcției 39](#_Toc107332723)

[Figură 4‑13 - Definiția unei gramatici prin intermediul fișierului în format JSON 40](#_Toc107332724)

[Figură 4‑14 - Transpunerea variabilelor globale din fișierul producător în fișierul cu gramatica 41](#_Toc107332725)

[Figură 4‑15 - Definiția gramaticii pentru o instrucțiune condițională și a elementelor constitutive asociate 41](#_Toc107332726)

[Figură 4‑16 - Arborele de sintaxă generat pentru o instrucțiune condițională, folosind abordarea top-down 42](#_Toc107332727)

[Figură 4‑17 - Construcția unei definiții terminale a unei instrucțiuni condiționale, folosind abordarea bottom-up 43](#_Toc107332728)

[Figură 4‑18 - Regulile de producție asociate unei declarații de variabilă 43](#_Toc107332729)

[Figură 4‑19 - Regulile de producție asociate unor instrucțiuni 44](#_Toc107332730)

[Figură 4‑20 - Regulile de producție asociate unor expresii 44](#_Toc107332731)

[Figură 4‑21 - Regulile de producție asociate unei instrucțiuni iterative din cadrul unei bucle 44](#_Toc107332732)

[Figură 4‑22 - Definiția gradului de unicitate 45](#_Toc107332733)

[Figură 4‑23 - Customizarea fișierului producător 46](#_Toc107332734)

[Figură 4‑24 - Fișier de configurare 46](#_Toc107332735)

# Listă tabele

[Tabel 3‑1 - Definiția direcționată de sintaxă a formei postfixe a unei expresii 21](#_Toc107332736)

[Tabel 4‑1 - Elemente descriptive ale API-ului platformei de testare 29](#_Toc107332737)

# Listă de acronime

AIG Automatic Item Generation

API Application Programming Interface

AST Abstract Syntax Tree

AQG Automatic Question Generation

CRUD Create Read Update Delete

CSS Cascading Style Sheets

DBMS Database Management System

DI Dependency Injection

DOM Document Object Model

DTO Data Transfer Object

EFC Entity Framework Core

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

JS Javascript

MVC Model View Controller

ORM Object Relational Mapper

JSON Javascript Object Notation

JWT JSON Web Token

SDT Syntax Directed Translation

SPA Single Page Application

# Introducere

În procesul educațional, testarea acumulării de cunoștințe prin întrebări este una dintre metodele fundamentale de evaluare. Acestea oferă beneficii precum stimularea memoriei pentru regăsirea și sedimentarea informației, focusarea pe conceptele importante din materialul studiat și în anumite circumstanțe obținerea de feedback atunci când există o nelămurire.

Ca o consecință a dezvoltării rapide și a nevoii de personal tot mai mari în industria tehnologiei informației, formarea unui număr mare de specialiști devine o provocare pentru resursa umană limitată din mediul universitar.

Profesorii se confruntă în mod regulat cu sarcina de a crea itemi pentru testarea cunoștințelor din cauza publicării materialelor existente și accesului deschis la acestea, iar compunerea manuală a întrebărilor poate deveni un proces complex ce necesită experiență, resurse și timp. Astfel, apare nevoia de a crea într-un timp scurt un set amplu de exerciții.

Printre sarcinile comune pe care profesorii trebuie să le îndeplinească se numără și corectarea testelor sau oferirea de feedback asupra rezolvărilor propuse de către studenți.

Caracterul repetitiv al sarcinilor expuse anterior subliniază importanța rezolvării problemei și impactul puternic pe care l-ar avea o soluție automatizată, precum reducerea resurselor necesare pentru producerea unui test cât și maximizarea timpului care poate fi dedicat de către profesori pentru sarcini mult mai importante.

În momentul de față soluțiile cele mai cunoscute și utilizate de evaluare automată au reușit să degreveze cadrele didactice de partea de corectare, folosind întrebările și răspunsurile furnizate de către acestea, fiind pretabile pentru orice domeniu. Pentru a preveni plagiatul, este necesară versatilitatea și complexitatea itemilor din cadrul testelor, iar din acest punct de vedere soluțiile menționate anterior necesită o bancă mare de itemi. Există și soluții particulare care au reușit să degreveze cadrele didactice de partea de creare de itemi și generare a răspunsurilor, precum și să asigure o versatilitate a itemilor, utilizând șabloane parametrizate ale întrebărilor și algoritmi specifici care să genereze răspunsul. Acestea nu mai sunt, în schimb, pretabile oricărui domeniu, iar utilizarea unor șabloane, deși previne plagierea, nu asigură întotdeauna înțelegerea materiei, rezolvarea acestora putând fi învățată mecanic.

Se poate observa că generarea procedurală are un rol important în cadrul automatizării sarcinii de creare a itemilor. Aceasta este una din ramurile creării de conținut multimedia, cunoscută pentru producerea, manipularea și modificarea datelor prin mijloace automate și se bazează pe fuziunea dintre conținutul furnizat de om și aleatorismul utilizat în cadrul algoritmilor[[1]](#footnote-1).

O constrângere în domeniul generării automate de întrebări este reprezentată de faptul că nu poate fi construită o soluție universal valabilă. Odată stabilite constrângerile particulare ale unui generator destinat unui domeniu, șabloanele parametrizate pot asigura o oarecare versatilitate a itemilor, însă nu asigură înțelegerea acestora de către student. Prin urmare, problema dificilă care trebuie rezolvată constă în furnizarea unui grad și mai mare de versatilitate.

Lucrarea de față propune o soluție la această problemă, țintind domeniul programării. Proiectul a fost conceput cu scopul de a dezvolta un sistem de testare automatizat și de a inova modalitatea de evaluare din cadrul cursului de Structuri de Date și Algoritmi. Predarea programării se confruntă cu probleme specifice, cum ar fi înțelegerea conceptelor de programare, dar și a algoritmilor pentru rezolvarea sarcinilor de programare. În cadrul cursurilor introductive, studenții încearcă să evite înțelegerea conceptelor urmând niște scurtături, cum ar fi învățarea codului prin memorare sau copierea programelor de la colegi.

Prin urmare, se profilează obiectivul principal pe care generatoarele de exerciții de programare trebuie să îl îndeplinească: furnizarea varietății necesare de exerciții pentru studenți astfel încât să le testeze abilitățile practice de-a lungul cursului și să rezolve problema plagiatului. Pentru a ajuta la rezolvarea problemei de scalabilitate a cursurilor, generatoarele de exerciții ar trebui să îndeplinească și alte obiective adiționale. O varietate de exerciții nu asigură faptul că acestea reflectă conceptele și tehnicile de programare urmărite în evaluarea cunoștințelor. Prin urmare, trebuie asigurată și îndeplinirea acestui obiectiv. De asemenea, testarea corectitudinii soluției studentului și furnizarea de feedback în mod automat reprezintă alte modalități prin care profesorul poate fi degrevat de o sarcină repetitivă și niște componente necesare în cadrul unui sistem de evaluare.

Există o multitudine de tipuri de exerciții care pot fi folosite pentru a evalua cunoștințele de programare ale unui student. În cadrul lucrării au fost abordate exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi, precum și exercițiile semantice ce constau în înțelegerea fluxului unui program.

Pentru primul tip de exerciții soluția propusă constă în furnizarea tiparului întrebării, a unui fișier de configurare cu ajutorul căruia se determină termenii ce vor fi înlocuiți în cadrul acesteia cu datele generate și un fișier producător care încapsulează algoritmul vizat de către întrebare, care compilat și rulat va genera datele întrebării și rezultatul aferent.

Al doilea tip de exerciții a necesitat o soluție mai complexă deoarece o parte din întrebare constă dintr-o secvență de cod care trebuie să reflecte conceptele urmărite în evaluarea cunoștințelor. Pentru a produce acel cod a fost indispensabilă o abordare originală asupra unor concepte de design a compilatoarelor precum gramaticile sau tabela de simboluri. Spre deosebire de tipul precedent de exerciții acesta utilizează și un fișier cu gramatica folosită pentru generarea secvenței de cod. O altă diferență constă în faptul că fișierul producător este inițial incomplet și necesită inserarea secvenței de cod generate pentru a putea fi compilat și rulat.

Întrebarea și fișierele sunt introduse în cadrul platformei prin intermediul unei interfețe administrator. Ulterior, profesorul poate crea un test care să cuprindă întrebarea și să poată fi susținut de către student. O imagine mai detaliată asupra întregului sistem este furnizată în capitolele următoare.

Primul capitol prezintă succint importanţa temei, utilitatea unui astfel de sistem, precum și obiectivele lucrării ce se doresc a fi îndeplinite prin elaborarea proiectului. De asemenea, este furnizată o imagine de ansamblu asupra conținutului acestei lucrări și a modului în care aceasta este structurată alături de o prezentare sumară a rezultatelor obţinute.

Capitolul al doilea își propune să ofere prin intermediul literaturii relevante recente o perspectivă generală asupra contextului actual în care se află domeniul generării procedurale de întrebări și o analiză detaliată a soluțiilor similare și a implementărilor existente, cu un focus pe exercițiile de programare. Adițional, acesta va expune API-uri și componente software relevante.

Cel de-al treilea capitol prezintă sumar etapele parcurse de-a lungul întregului proces de compilare pentru a furniza o perspectivă de ansamblu asupra conceptelor și contextului în care au fost utilizate și a modului de funcționare al acestora. Analiza sintactică, generarea de cod intermediar și tabela de simboluri constituie pilonii procesării de text și generării de cod pentru întrebările axate pe înțelegerea fluxului programelor din prezentul proiect.

Al patrulea capitol cuprinde detaliile privind implementarea practică precum definirea cerinţelor complete pentru sistemul propus, prezentarea cazurilor de utilizare, definirea arhitecturii și proiectarea sistemului, tehnologiile sau API-urile folosite pentru dezvoltarea acestuia, descrierea cazurilor de testare și elaborarea raportului de testare, precum și problemele întâmpinate pe parcursul întregului proces.

Capitolul cinci este dedicat prezentării rezultatelor experimentale, în urma măsurătorilor de performanță, iar ultimul capitol face o sinteză asupra principalelor idei din lucrare, furnizând direcţii și idei pentru continuarea acesteia.

# Literatură relevantă recentă

Pentru a reduce costurile asociate procesului manual de generare al întrebărilor și pentru a rezolva problema unui flux constant de conținut, au fost dezvoltate tehnici de generare procedurală ale acestora.

Acest capitol își propune să furnizeze o imagine de ansamblu asupra contextului actual în care se află domeniul generării procedurale de întrebări, o analiză detaliată a soluțiilor similare și a implementărilor existente, precum și dezavantajele fiecărei propuneri, punând accent pe cele care vizează exercițiile de programare.

În ciuda faptului că volumul cercetărilor științifice în domeniul generarării procedurale de întrebări a cunoscut o creștere în ultimii ani, munca privind generarea automată de exerciții pentru programare a fost relativ limitată în comparație cu celelalte subdomenii. Au apărut mai multe propuneri pentru dezvoltarea de software educațional specializat în a ajuta studenții să înțeleagă conceptele de bază de programare și să dezvolte abilități de rezolvare a problemelor. Soluțiile propuse în prezent se bazează în mare măsură pe tipare parametrizate și nu oferă gradul dorit de versatilitate și complexitate, augmentând astfel utilitatea prezentei lucrări.

## Metode generative pentru întrebările cu răspuns liber

Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler și Salam Al-Emari [1] au analizat peste 90 de lucrări științifice legate de generarea întrebărilor în domeniul educațional, furnizând o imagine de ansamblu asupra comunității AQG, direcțiile curente și progresele realizate. Acesta prezintă etapele procesului generator și metodele generative pentru diferite tipuri de întrebări printre care se numără și întrebările cu răspuns liber vizate în cadrul proiectului, precum și modalitățile prin care se poate face evaluarea calității itemilor.

În domeniul evaluării automate există mai multe categorii de întrebări care pot fi utilizate în cadrul testelor. Exercițiile care pot fi folosite cu precădere în domeniul programării pot fi împărțite în trei categorii de bază: întrebări cu răspunsuri multiple, întrebări cu răspuns liber și sarcini de programare. Simplitatea întrebărilor cu alegere multiplă le-a făcut foarte populare în sistemele de management al învățării, cum ar fi Moodle. Cu toate acestea, în predarea programării, întrebările cu răspunsuri multiple pot fi utile doar pentru adoptarea unor elemente teoretice de bază, nu și pentru dobândirea de abilități practice în rezolvarea sarcinilor de programare. De aceea, proiectul curent este focusat exclusiv pe întrebările cu răspuns liber.

Alcătuirea întrebării (setul întrebare-răspuns corect) este principala sarcină a întregului proces de generare. În funcție de tipul acesteia și de formatul răspunsului sunt implicate diverse abordări. În cadrul clasificării metodelor de generare, metoda folosită în mod uzual pentru producerea întrebărilor cu răspuns liber este cea bazată pe șabloane ce constă în folosirea unei structuri de bază, ale cărei goluri sunt customizate cu valori care întrunesc caracteristicile sintactice sau semantice precizate. Șabloanele necesită intervenția umană prin construirea manuală a acestora, ceea ce face ca din perspectiva costului, această metodă să fie considerată scumpă. Un alt dezavantaj al folosirii acestei metode este dat de structură limitată a întrebărilor generate din punct de vedere al diversității lingvistice.

Din perspectiva costului nu a putut fi identificată o alternativă mai bună pentru generarea întrebărilor cu răspuns liber, din cauză că și restul metodelor prezentate au în acest moment un cost ridicat (ex. necesitatea unei cantități mari de date etichetate corespunzător). Metodele alternative nu au însă dezavantajul structurii rigide a întrebării. Deși acestea produc întrebări mult mai diversificate din punctul de vedere al topicii propoziției și al limbajului, nu sunt potrivite exercițiilor ce urmăresc să reflecte anumite concepte și tehnici de programare. Se justifică, astfel, folosirea uzuală a metodei șabloanelor în acest domeniu.

Ca practică generală, în urma generării, se realizează evaluarea calității întrebării. Criteriile de evaluare se orientează asupra calității lingvistice și educaționale. Din punct de vedere lingvistic se urmărește corectitudinea gramaticală, fluența, ambiguitatea semantică și lipsa erorilor. Din punct de vedere educațional se urmărește utilitatea întrebării, relevanța în cadrul domeniului și obiectivul didactic atins de subiectul evaluat prin răspunsul la acea întrebare. De asemenea, mai sunt și alte metrici relevante pentru calitate precum dificultatea sau nivelul cognitiv țintă. Determinarea dificultății întrebării se poate face prin determinarea numărului de persoane evaluate care au răspuns corect din totalul celor examinați, compararea dificultății prezise cu performanțele obținute de studenți sau prin utilizarea unor mecanisme de rezolvare automată.

Ținând cont de observațiile făcute de către autori pe baza analizei compilației de lucrări științifice, implementarea acestui proiect folosește la bază ca metodă de generare șabloanele. Prin utilizarea acestei metode, întrebările întrunesc criteriile de calitate menționate anterior deoarece șablonul trece prin filtrul unei persoane cu experiența necesară a-l întocmi.

## Soluții ce utilizează șabloane parametrizate

Generatorul de exerciții de programare implementat în Python de Danijel Radošević, Tihomir Orehovački și Zlatko Stapić [2], se bazează pe o listă de specificații, un fișier de configurare și tipare de cod în limbajul C++.

Lista de specificații conține elemente de tipul atribut:valoare, fiind constituită inițial din numele fișierului de output. Aceasta este ulterior modificată prin adăugarea mai multor elemente alese aleator din liste predefinite ce conțin nume de fișiere șablon sau diverse denumiri pentru identificatorii variabilelor. Lista finală de specificații conține atribute precum fișierul de output, numele studentului, ID-ul studentului, tiparul principal și alți parametrii specifici exercițiului.

Autorii lucrării menționează că au definit mai multe tipuri de tipare: un tipar cu codul principal al programului și cu exercițiile sub formă de comentarii și tipare cu dimensiuni mai mici ce conțin secvențe de cod. Exemplul furnizat de aceștia arată o posibilă structură pentru tiparul principal.

În cadrul procesului generativ, porțiunile delimitate de caracterul „#” din tipare sunt înlocuite prin intermediul valorilor asociate din fișierul de configurare, care referențiază fie un atribut din lista de specificații fie numele unui fișier cu tiparul de dimensiune mai mică. Înlocuirea se face, astfel, fie prin obținerea directă a valorii atributului din lista de specificații fie prin generarea tiparului menționat.

O problemă a acestei soluții sunt cerințele relativ mari pentru profesori, din cauză că fiecare exercițiu necesită un set particular de tipare. Deși acestea ar putea fi moștenite parțial de la exercițiile anterioare, iar configurația generatorului ar putea fi, de asemenea, moștenită parțial, întregul proces necesită un control granular asupra mulțimii de fișiere, ceea ce nu este fezabil pentru o diversitate de tipuri de exerciții. În implementarea proiectului, această problemă a multitudinii de tipare ce trebuie întocmite, a fost rezolvată prin generarea unei diversități de secvențe de cod prin definirea unei gramatici.

O altă problemă este lipsa furnizării unei soluții automate de notare a exercițiilor și metoda de furnizare a acestora prin comentarii. Acest aspect necesită timp suplimentar din partea profesorilor și ar putea fi îmbunătățit. Prin urmare, lucrarea de față a definit în procesul generator, un mecanism de producere a răspunsurilor, pentru ca ulterior soluția studentului să poată fi corectată în mod automat.

Akiyoshi Wakatani și Toshiyuki Maeda [3] au pornit de la premiza că la baza învățării unui limbaj de programare se află atât dobândirea cunoștințelor de gramatică a limbajului cât și capabilitatea de a înțelege fluxul unui program. Prin urmare, aceștia au pregătit două tipuri de exerciții: exerciții de sintaxă prin care cursanții să corecteze erorile programelor furnizate, și exerciții de semantică prin care cursanții să estimeze rezultatul programelor, fără a le executa. Acestea pot fi accesate prin intermediul unei aplicații web în limbajul PHP, care le generează automat folosind șabloane în limbajul C. Exercițiile generate sunt evaluate automat.

Pentru primul tip de exerciții, interfața grafică permite vizualizarea unui program cu erori și a descrierilor erorilor furnizate de compilator. Prin intermediul acesteia, cursanții corectează programul utilizând cunoștințele despre gramatica limbajului și mesajele de eroare până când toate erorile sunt complet eliminate.

Exercițiile sunt generate prin adăugarea de bug-uri unor programe șablon și prin înlocuirea numelor de variabile. Au fost definite trei metode de adăugare a erorilor la programul șablon. În primul rând, numele variabilelor sunt înlocuite cu valori alese aleator dintr-o listă predefinită de nume. Astfel, pot apărea în program variabile nedeclarate. A doua metodă se bazează pe scrierea greșită a unor șiruri de caractere definite în prealabil (de exemplu, stdio.h poate fi schimbat în stdi.h). A treia metodă înlocuiește un șir de caractere cu un alt cuvânt desemnat, pentru a genera o eroare întâlnită des (de exemplu, înlocuirea „;” cu „:”).

După ce un cursant corectează un program care are erori, scriptul PHP salvează programul corectat ca fișier cu un nume unic, îl compilează și trimite înapoi mesajul de la compilator către cursant. Șirul de caractere „OK” este trimis înapoi, dacă compilatorul nu produce niciun mesaj.

Autorii menționează că programele șablon sunt pregătite în conformitate cu progresul învățării și sunt necesare între 10 și 20 programe șablon pentru a produce o varietate de programe.

Al doilea tip de exerciții, cele de estimare a output-ului unui program, au scopul de a îmbunătăți capacitatea de înțelegere a programelor din punct de vedere semantic. Interfața grafică a aplicației web permite vizualizarea unui program care conține diverse instrucțiuni. Cursantul citește programul și estimează rezultatul funcției prin înțelegerea semanticii acestuia. După scrierea valorii estimate în zona de răspuns și apăsarea butonului de verificare, este afișată comparația răspunsului corect cu estimarea.

Autorii menționează că programele date pot include apeluri recursive de funcții și calcule complicate, ceea ce face mai dificil de estimat rezultatul. Pentru a crea programe corecte, este indispensabilă înțelegerea comportamentului programelor care includ instrucțiuni condiționale, bucle și apeluri de funcții. Prin urmare, aceștia susțin că aceste exerciții sunt utile pentru a îmbunătăți capacitatea cursanților de a înțelege programe. Prin schimbarea adecvată a numelor variabilelor și a valorilor constantelor și menținând lungimea programului suficient de compactă pentru estimare, cursanții pot efectua exerciții în mod repetat pentru a spori eficacitatea educației. Parametrii constanți și modelele de calcul trebuie să fie date corect, deoarece parametrii greșiți pot provoca excepții precum împărțirea la zero sau erori de acces la memorie.

Justificarea obiectivelor didactice atinse de către exercițiile menționate, le fac pretabile pentru a fi produse de către generatorul implementat în cadrul proiectului. Ținând cont de faptul că primul tip vizează noțiuni de bază introductive și nu reprezintă o provocare pentru studenții acomodați cu un limbaj de programare, a fost ales doar al doilea tip de exerciții. Pentru acesta, soluția curentă furnizează o gamă largă de secvențe de cod unice fără a mai fi necesară definirea a 10-20 de tipare.

## Soluții ce utilizează noduri de instrucțiuni

Deși întrebările parametrizate pot fi utile pentru a preveni plagiatul, ele necesită destul de mult timp din partea cadrelor didactice pentru crearea tiparelor. Un potențial răspuns la această problemă este furnizat de Thomas James Tiam-Lee și Kaoru Sumi [4] care prezintă o abordare pentru generarea procedurală a exercițiilor de programare fără a utiliza șabloane parametrizate. Pentru a reprezenta exercițiile într-o manieră structurată, fiecare exercițiu este tratat ca o succesiune de instrucțiuni pe care elevul trebuie să le efectueze. Fiecare instrucțiune este reprezentată de un singur nod, care poate fi asociat cu o operație, o condiție sau returnarea unei valori. Prin urmare, exercițiul este o colecție de noduri, care va fi parcursă succesiv.

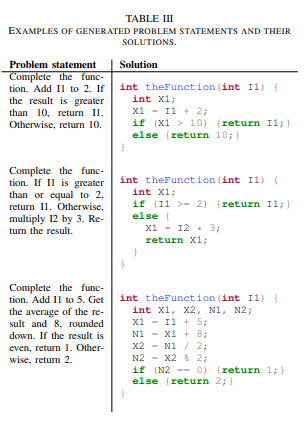
Un nod de operație, se referă la o singură operație aritmetică și este urmat întotdeauna de un singur nod. Un nod de condiție, se referă la o singură expresie condițională și este întotdeauna urmat de două noduri, unul pentru când condiția este adevărată și unul pentru când condiția este falsă. Un nod de returnare, se referă la un punct final al funcției și nu este urmat de un alt nod. Operații de nivel superior pot fi adăugate prin abstractizarea lor ca blocuri de operații sau de condiții. Blocurile constau într-o secvență internă de operații elementare.

Pornind de la o valoare definită a complexității codului dorit, se generează o structură aleatorie. Complexitatea unei structuri este o metrică definită ca numărul de noduri asociate cu o operație sau cu o condiție. Această definiție se bazează pe ideea că un exercițiu devine mai complex pe măsură ce crește numărul de instrucțiuni necesare pentru a-l finaliza.

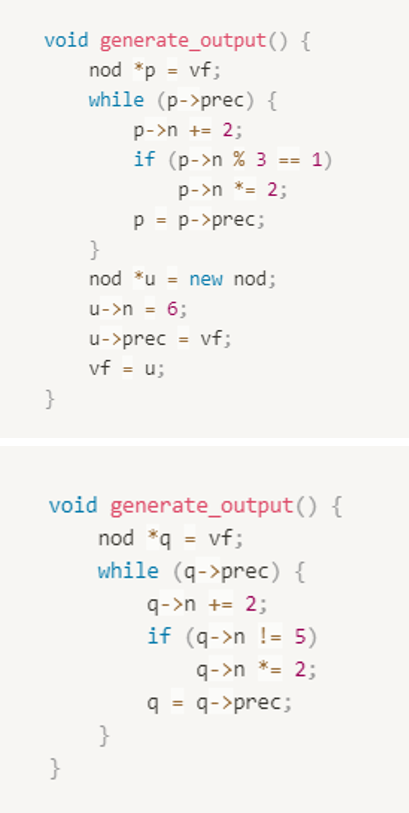
Primul nod este inițializat fie ca nod de operație fie ca nod de condiție. Apoi, într-unul din locurile disponibile din cadrul structurii este atașat aleatoriu, în mod asemănător, fie un nod de operare, fie un nod de condiție. Această operațiune este efectuată până când structura conține un număr de noduri egal cu complexitatea. În locurile libere rămase, sunt atașate nodurile de returnare.

După definirea structurii exercițiului, parametrii fiecărui nod sunt asignați utilizând un algoritm ce a luat în calcul considerațiile detaliate în cadrul lucrării. Odată construită secvența de instrucțiuni, fiecare nod este parcurs și în funcție de tipul acestuia este generat atât un text în limbaj natural pentru crearea exercițiului, cât și codul Java asociat pentru a evalua soluția elevului. Pentru crearea cerinței, există mai multe texte predefinite pentru fiecare tip de nod, iar din acestea este ales unul, în mod aleator. Ulterior, porțiunile marcate goale sunt înlocuite cu variabile și parametrii din structura nodului.

În acest studiu au fost folosite doar operații simple de calcul pentru producerea exercițiilor de programare precum operațiile aritmetice (de exemplu, adunarea, înmulțirea), operațiile condiționale (de exemplu, mai mare, mai mic) și câteva blocuri. De asemenea, tipurile de date sunt limitate doar la numere întregi. Din acest motiv, exercițiile generate, în ciuda faptului că nu sunt derivate dintr-un șablon parametrizat, arată în continuare similare între ele.



Figură 2‑1 - Secvențe de cod generate de James Tiam-Lee și Kaoru Sumi



Figură 2‑2 - Secvențe de cod generate în cadrul acestui proiect

Prin urmare, soluția propusă în lucrarea de față și-a propus și a reușit să obțină secvențe de cod mai complexe care conțin și bucle și care nu se limitează doar la un anumit tip de date. De asemenea, ideea că un exercițiu devine mai complex pe măsură ce crește numărul de instrucțiuni necesare pentru a-l finaliza a contribuit la definirea metricii asociate complexității folosite în cadrul proiectului.

## Soluții ce utilizează gramatici (biblioteca Tracery.js)

Generarea procedurală este utilizată într-un spectru larg de domenii, însă cel mai popular dintre acestea este cel al jocurilor, unde texturile, efectele sonore și povestea sunt produse cu ajutorul gramaticilor.

O gramaticăeste constituită dintr-un set de simboluri terminale (simbolurile elementare ale limbajului definit de gramatică), un set de neterminale (un neterminal reprezintă un șir de terminale), un set de reguli de producție (fiecare regulă de producție constă dintr-un neterminal, o săgeată și o secvență de terminale și/sau neterminale) și un simbol de start desemnat din rândul neterminalelor. Aceasta definește sintaxa unui limbaj.

Această metodă de generare nu este comun folosită în domeniul educațional. Pornind de la faptul că întrebările sunt de fapt un text, iar gramaticile pot produce o gamă diversificată de texte utilizând un set de reguli de producție, este foarte probabil ca această metodă să se preteze domeniului. De asemenea, pentru o particularizare a întrebărilor de natură generală la întrebări ce vizează exerciții de programare, această abordare e o opțiune la fel de viabilă. Astfel, utilizând această euristică, lucrarea de față și-a propus să obțină întrebări cu un grad mare de versatilitate prin intermediul gramaticilor.

Tracery [5] este un instrument generativ de text conceput cu scopul de a fi folosit de către autori începători și experți sau de către persoane care nu se pot autoidentifica drept „programatori” pentru crearea unor combinații surprinzătoare de text în mod algoritmic. În practică, această bibliotecă a fost utilizată pentru crearea unei game largi de povești, poezii, dialoguri și chiar imagini în format SVG sau pagini HTML cu gramatici profund imbricate.

Utilizarea șabloanelor și gramaticilor pentru a crea text structurat, dar variabil în același timp reprezintă o tehnică comună în generarea de povești. Generatorul primește la intrare o gramatică formală scrisă ca obiect JSON, ce conține simbolurile și regulile de expandare asociate acestora. Aceasta va trece printr-un sistem modular de părți interfuncționale: un parsator și un motor de expandare. Prin înlocuirea treptată a simbolurilor cu una din regulile aferente acestora aleasă în mod aleator, se asigură variabilitatea în timp și se ajunge la forma finală a textului. Sintaxa limbajului Tracery este simplă și lizibilă, permițând utilizatorului să definească o gramatică în limbaj natural. Pentru a semnala simbolurile care vor fi extinse recursiv sunt folosite hashtag-uri.

Această tehnică a fost îmbunătățită cu ajutorul unui mecanism care generează text într-un mod mai riguros prin utilizarea unor funcții care pot fi aplicate după ce un simbol este expandat. Aceste funcții sunt numite modificatori. Astfel, au fost rezolvate problemele generative comune precum pluralizarea, conjugarea sau scrierea cu literă mare.

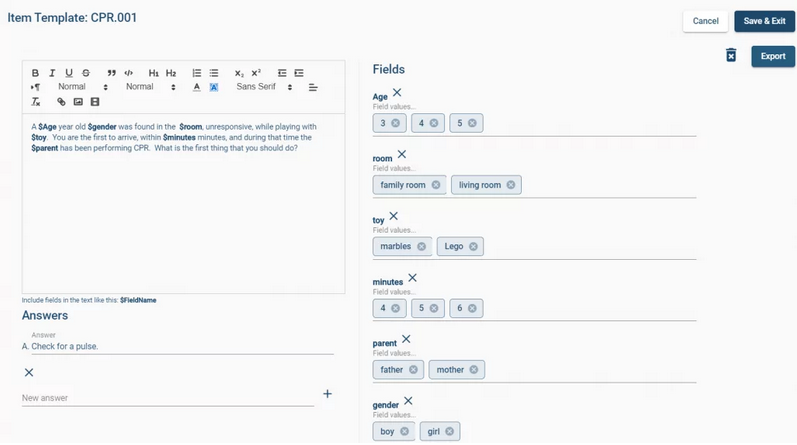
Cu această structură de date simplistă, autorii pot produce un text structurat și interesant din prisma faptului că poate fi lipsit pe alocuri de logică, însă în cazul exercițiilor de programare, acest lucru nu este de dorit. Deși biblioteca prezentată se pretează generării de text, nu permite un control mai fin al procesului generator pentru a putea produce un cod fără erori. Prin urmare, proiectul a implementat un generator bazat pe conceptele menționate anterior, adăugând un set de filtre care să verifice dacă o instrucțiune are sens în raport cu instrucțiunile generate anterior.

## Soluții comerciale similare (ASC, Brio)

Există mai multe instrumente folosite în procesul de evaluare, variind de la instrumente folosite doar de anumite universități până la instrumente care au fost dezvoltate pentru uz comercial.

Platforma de testare ASC [6] utilizează generarea automată de itemi (AIG), încercând să automatizeze o parte din efortul implicat în crearea articolelor, din cauză că acesta este unul dintre aspectele cele mai consumatoare de timp ale dezvoltării unui test.

Tehnologia curentă de generare automată a unei game largi de itemi se bazează pe șabloane. Utilizatorul marchează prin intermediul simbolului $ porțiunile din text care sunt inserabile și definește pentru fiecare porțiune niște valori ce vor fi folosite în momentul înlocuirii. Algoritmul va produce toate permutările posibile ale șablonului. Itemii rezultați nu sunt mutați automat în banca de itemi, ci sunt revizuiți de persoane autorizate, din motive precum crearea de scenarii puțin probabile sau irelevante. Cu toate acestea, eficiența generală în procesul de creare al întrebărilor s-a îmbunătățit considerabil, iar itemii generați sunt de o calitate mai bună decât itemii scriși în mod tradițional, deoarece procesul i-a făcut pe autori să se gândească mai profund la ceea ce evaluează și cum. Platforma nu recomandă utilizarea mai multor instanțe ale aceluiași tip de item generat procedural în cadrul unui formular de testare.



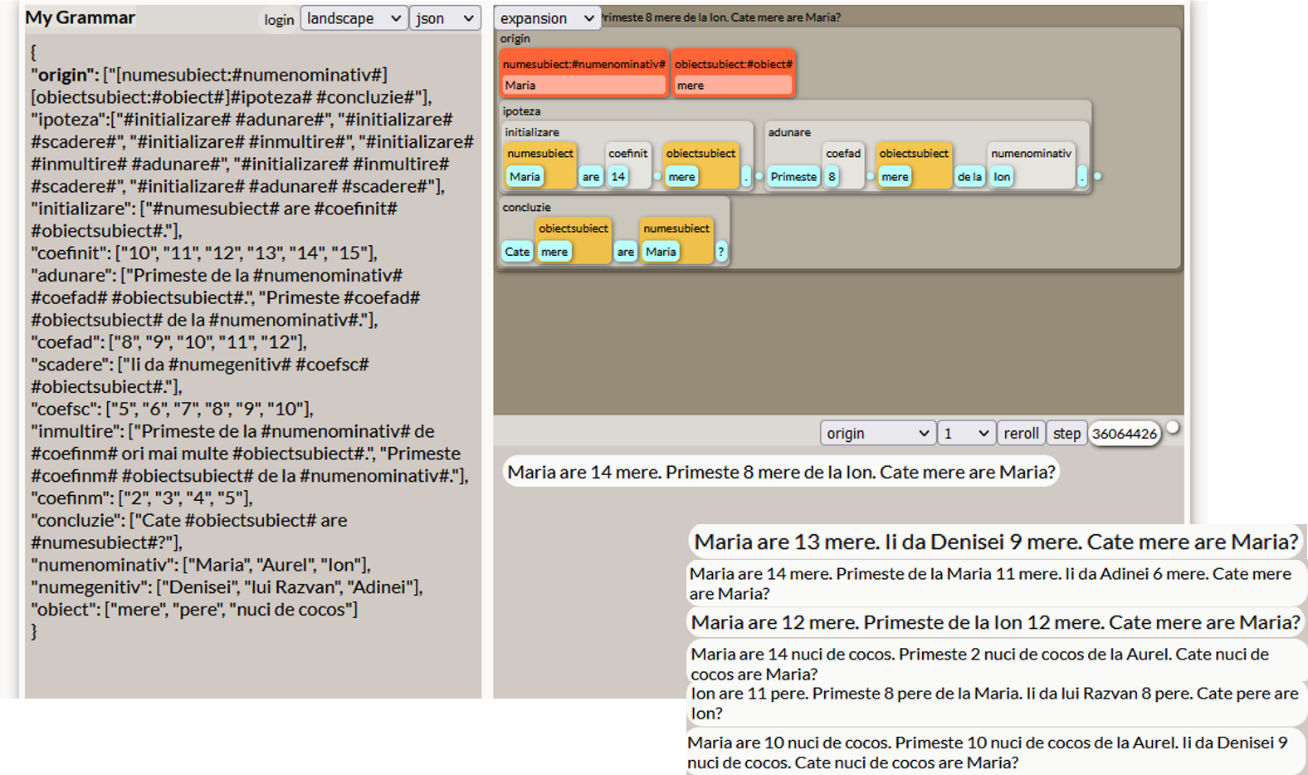
Figură 2‑3 - Interfața de creare a unor itemi ce vor fi generați automat în cadrul platformei ASC

După cum se poate observa în Figura 2-3, un dezavantaj al platformei este faptul că răspunsurile trebuie furnizate de către autorul întrebării, problemă care a fost adresată în cadrul proiectului curent. De asemenea, utilizarea unor șabloane parametrizate care au o structură rigidă, nu are avantajele utilizării unei gramatici ce are atât o structură rigidă prin regulile ei, dar și variabilitate prin modul aleator de alegere al regulilor.

Platforma de testare Brio [7] furnizează elevilor din ciclul primar (I-IV) teste de antrenament bazate pe generarea programatică, dinamică și randomizată de itemi. Această platformă deține un număr foarte mare de itemi calibrați prin testarea pe eșantioane reprezentative de elevi români, garantând acoperirea minuțioasă a întregii programe școlare anuale, în cadrul fiecărui test.



Figură 2‑4 - Itemii generați automat pentru un test în cadrul platformei Brio



Figură 2‑5 - Itemi similiari generați cu ajutorul unei gramatici prin intermediul instrumentului online Tracery

În ciuda faptului că platforma Brio generează automat atât itemii cât și răspunsurile, tiparele întrebărilor sunt destul de simpliste. Deși nu este precizată explicit, metoda utilizată pentru generare pare a fi bazată pe șabloane. În acest caz, o gramatică ar permite obținerea unor soluții mai versatile din punctul de vedere al cerinței. De asemenea, în loc să fie construite o mulțime de tipare, o gramatică permite generarea mai multor tipare prin simpla definire a ei.

# Noțiuni generale de funcționare a unui compilator

Modelele, teoria și algoritmii asociați unui compilator, explicate în detaliu de Alfred Aho, Monica Lam, Ravi Sethi, și Jeffrey Ullman [8], pot fi aplicate unei game largi de probleme în design-ul și dezvoltarea de software. Prin urmare, prezentarea sumară a etapelor parcurse de-a lungul întregului proces de compilare poate furniza o perspectivă de ansamblu asupra conceptelor și contextului în care au fost utilizate și a modului de funcționare al acestora. Analiza sintactică, generarea de cod intermediar și tabela de simboluri constituie pilonii procesării de text și generării de cod pentru întrebările axate pe înțelegerea fluxului programelor din prezentul proiect.

## Etapele procesului de compilare

**Faza de analiză** a unui compilator constă în descompunerea codului sursă în părțile constitutive ale limbajului de programare, impunerea unei structuri gramaticale și crearea unei reprezentări intermediare asociate codului. O structură gramaticală implică prezența unor reguli de sintaxă specifice limbajului. **Sintaxa** unui limbaj de programare descrie forma adecvată a programelor sale, în timp ce **semantica** limbajului definește sensul acestora, mai precis, ce face fiecare program atunci când este executat. Pentru specificarea sintaxei se folosește o notație utilizată pe scară largă, numită gramatică independentă de context[[2]](#footnote-2).

Sarcina principală a **analizatorului lexical** este să citească caracterele codului sursă, să le grupeze în lexeme (șiruri de caractere) și să producă o secvență de tokeni formați din tipul lexemului și opțional, un pointer către intrarea din tabela de simboluri. Atunci când analizatorul lexical descoperă un lexem care constituie un identificator (nume de constante, variabile sau funcții), introduce acel lexem în tabelă. Tokenii sunt trimiși către parsator pentru analiza de sintaxă[[3]](#footnote-3).

**Parsatorul** obține un șir de tokeni de la analizatorul lexical și verifică faptul că șirul de tokeni poate fi generat de gramatica limbajului. Parsatorul va raporta orice erori de sintaxă într-un mod inteligibil. Pentru programele bine formate, analizatorul construiește un arbore de parsare și îl transmite restului compilatorului pentru procesare ulterioară.

Fazele ulterioare ale compilatorului trebuie să analizeze rezultatul parsatorului, pentru a asigura conformitatea programului cu regulile care nu sunt verificate de parsator[[4]](#footnote-4). O gramatică nu distinge între identificatorii care sunt șiruri de caractere diferite. În schimb, toți identificatorii sunt reprezentați printr-un simbol precum id în gramatică. Într-un compilator pentru un astfel de limbaj, faza de **analiză semantică** verifică dacă identificatorii sunt declarați înainte de a fi utilizați[[5]](#footnote-5).

Dacă partea de analiză detectează pe parcursul **verificării statice** (verificare făcută de compilator) incorectitudinea codului sursă din punct de vedere sintactic sau semantic (nu sunt urmate regulile sintactice și semantice ale limbajului în care a fost scris), atunci trebuie să furnizeze mesaje informative, astfel încât utilizatorul să poată lua măsuri corective.

**Verificarea sintactică** nu verifică doar dacă programul respectă gramatica limbajului. De exemplu, constrângerile precum declararea unui identificator cel mult o dată într-un domeniu de valabilitate, sau încadrarea obligatorie a instrucțiunii break într-o buclă sunt de natură sintactică, deși nu sunt impuse de o gramatică utilizată pentru parsare.

**Verificările de tip** ale unui limbaj asigură că operatorii sau funcțiile sunt aplicate numărului și tipului corect de operanzi. Dacă este necesară conversia între tipuri (de exemplu, atunci când un număr de tip întreg este adăugat la un număr de tip float), atunci verificatorul de tip poate insera un operator în arborele de sintaxă pentru a reprezenta acea conversie[[6]](#footnote-6). În detectarea erorilor, au un rol esențial atât gramatica limbajului cât și tabela de simboluri[[7]](#footnote-7).

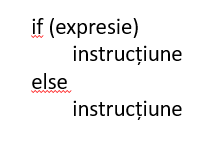
Pe parcursul etapei de analiză sunt colectate și stocate informații despre programul sursă într-o structură de date numită **tabelă de simboluri**, care este transmisă împreună cu reprezentarea intermediară către etapa de sinteză[[8]](#footnote-8).

**Faza de sinteză** translatează programul intermediar în programul țintă, utilizând informațiile din tabela de simboluri.

Partea de analiză este adesea numită partea de frontend a compilatorului, iar partea de sinteză este numită partea de backend.

## Analiza sintactică

Implicit, prin design, fiecare limbaj de programare are reguli precise care descriu structura sintactică a programelor corecte (de exemplu, în C, un program este format din funcții, o funcție din declarații și instrucțiuni, o instrucțiune din expresii și așa mai departe)[[9]](#footnote-9). Structură ierarhică a elementelor constitutive ale unui limbaj de programare este descrisă de o gramatică. De exemplu, o instrucțiune if-else poate avea forma din Figura 3-1.



Figură 3‑1 - Structura unei instrucțiuni if-else

Această instrucțiune este reprezentată de concatenarea cuvântului cheie if, o paranteză deschisă, o expresie, o paranteză închisă, o instrucțiune, cuvântul cheie else și o altă instrucțiune. Folosind variabila expr pentru a desemna o expresie și variabila stmt pentru a desemna o instrucțiune, această regulă de poate fi exprimată sub forma din Figura 3-2.



Figură 3‑2 - Regula de producție pentru o instrucțiune if-else

O astfel de regulă se numește o regulă de producție. Într-o regulă de producție, elementele lexicale precum cuvântul cheie if și parantezele sunt numite terminale. Variabilele precum expr și stmt sunt numite neterminale[[10]](#footnote-10).

O **gramatică independentă de context** are patru componente:

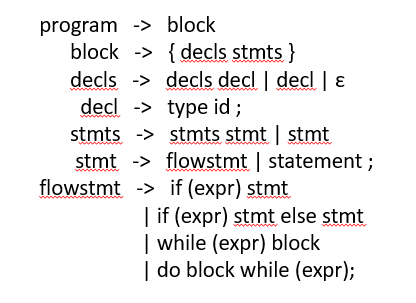
1. Un set de simboluri terminale, reprezentate de simbolurile elementare ale limbajului definit de gramatică;

2. Un set de neterminale, unde un neterminal reprezintă un șir de terminale;

3. Un set de reguli de producție, unde fiecare regulă de producție constă dintr-un neterminal (partea stângă a regulii de producție), o săgeată și o secvență de terminale și/sau neterminale (partea dreaptă a regulii de producție);

4. Un simbol de start desemnat din rândul neterminalelor.

O specificare a unei gramatici se face prin enumerarea regulilor de producție, începând cu regula de producție a simbolului de start. Cifrele, semnele (de exemplu, <, >, =) și cuvintele cheie (de exemplu, while) sunt terminale. Regulile de producție asociate aceluiași neterminal pot fi grupate prin simbolul |, cunoscut în mod convențional drept ,,sau’’[[11]](#footnote-11).



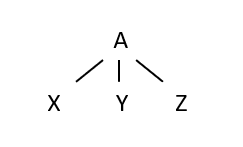
Figură 3‑3 - Gramatica unui limbaj de programare primitiv ce generează un set limitat de instrucțiuni

**Derivarea** unor șiruri de caractere pe baza unei gramatici se face începând cu simbolul de start și înlocuind în mod repetat un neterminal din regula de producție a simbolului cu corpul unei reguli de producție asociate acelui neterminal (un neterminal poate avea mai multe reguli de producție asociate). Șirurile terminale care pot fi derivate din simbolul de start formează limbajul definit de gramatică[[12]](#footnote-12).

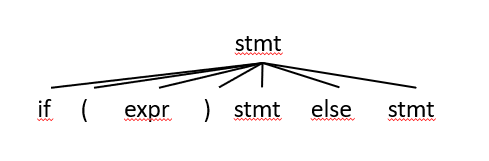
**Parsarea (analiza sintactică)** este definită de aflarea modului în care un șir de terminale este derivat din simbolul de start al gramaticii, iar în cazul în care șirul de terminale nu poate fi derivat din simbolul de start, de raportarea erorilor de sintaxă.

Un **arbore de parsare (arbore de sintaxă abstract)** este o imagine a unei derivări, aratând cum simbolul de start al unei gramatici derivă un șir în limbajul definit de gramatică. Fiecare nod dintr-un arbore de sintaxă reprezintă un element constitutiv al limbajului[[13]](#footnote-13).

Dacă neterminalul A are asociată o regulă de producție A -> XYZ, arborele de parsare are un nod interior etichetat cu A cu trei copii etichetați X, Y și Z, de la stânga la dreapta ca în Figura 3-4.



Figură 3‑4 - Arborele de sintaxă al neterminalului A



Figură 3‑5 - Arborele de sintaxă al unei instrucțiuni if-else

Având în vedere o gramatică independentă de context, un arbore de parsare este un arbore cu următoarele proprietăți:

1. Rădăcina este simbolul de start;

2. Fiecare frunză este un terminal;

3. Fiecare nod interior este un neterminal[[14]](#footnote-14).

Construcția de sus în jos a unui arbore de parsare ca cel din Figura 3-5, se face pornind de la rădăcină, și efectuând în mod repetat următorii doi pași:

1. Pentru nodul N se va selecta una dintre regulile de producție asociate și vor fi construiți copiii pentru simbolurile din corpul regulii de producție;

2. Se va găsi următorul nod la care urmează să fie construit un subarbore (de obicei, cel mai din stânga neterminal neexpandat al arborelui)[[15]](#footnote-15).

## Generarea de cod intermediar

Ideea de a asocia diverse proprietăți cu părțile constitutive ale unui limbaj de programare (de exemplu, valori și tipuri cu expresii) poate fi exprimată în cadrul gramaticilor. Neterminalele și terminalele pot avea asociate **atribute sintetizate**, iar regulile de producție pot avea asociate **reguli semantice** care descriu modul în care atributele sunt calculate la nivelul nodurilor arborelui de parsare unde acea regulă de producție este folosită.

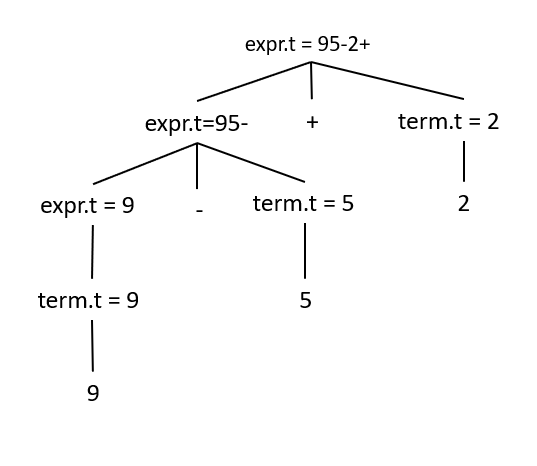
Reprezentarea intermediară a codului se realizează cu ajutorul arborilor de sintaxă abstractă ce modelează structura sintactică ierarhică a codului sursă[[16]](#footnote-16). Odată ce arborele de sintaxă este construit, codul intermediar poate fi generat prin evaluarea atributelor și executarea fragmentelor de cod asociate nodurilor din arbore[[17]](#footnote-17).

Se spune că un atribut este sintetizat dacă valoarea lui la un nod al arborelui analizat N este determinată de valorile atributelor copiilor lui N și N însuși. Atributele sintetizate au proprietatea că pot fi evaluate în timpul unei singure traversări de jos în sus a unui arbore de parsare[[18]](#footnote-18).

Fragmentele de cod aferente regulilor semantice sunt executate atunci când regula de producție este utilizată în timpul analizei de sintaxă și realizează **translatarea direcționată de sintaxă**[[19]](#footnote-19). Procesul de evaluare a valorii atributului, se numește **definiție direcționată de sintaxă**.

Pentru un șir de intrare dat, este construit arborele de parsare, iar apoi sunt aplicate regulile semantice pentru a evalua atributele fiecărui nod din arbore.

De exemplu, Figura 3-6 prezintă un arbore de parsare adnotat pentru 9-5+2. Valoarea 95-2+ a atributului de la rădăcină este notația postfixă pentru 9-5+2.



Figură 3‑6 - Valorile atributelor nodurilor arborelui de sintaxă unei expresii

Fiecare neterminal (expr, term) are un atribut t sub forma unui șir de caractere care reprezintă notația postfixă pentru expresia generată de acel neterminal. Simbolul || din regula semantică este operatorul utilizat în mod convențional pentru concatenarea șirurilor.

Tabel 3‑1 - Definiția direcționată de sintaxă a formei postfixe a unei expresii

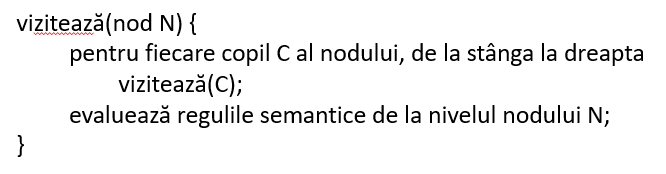
|  |  |
| --- | --- |
| REGULA DE PRODUCȚIE | REGULA SEMANTICĂ |
| expr -> expr1 + term | expr.t = expr1.t || term.t || ’+’ |
| expr -> expr1 - term | expr.t = expr1.t || term.t || ’-’ |
| expr -> term | expr.t = term.t |
| term -> 0 | term.t = ’0’ |
| term -> 1 | term.t = ’1’ |
| ... | ... |
| term -> 9 | term.t = ’9’ |

Forma postfixă a unei cifre este cifra însăși. De exemplu, regula semantică asociată cu regula de producție term -> 9 face ca term->t să fie 9 ori de câte ori această regulă de producție este utilizată la nivelul unui nod din arborele de parsare. Celelalte cifre sunt translatate similar. Când regula de producție expr -> expr1 + term este aplicată, valoarea term->t preia valoarea expr->t[[20]](#footnote-20).

Definiția direcționată de sintaxă din exemplul precedent are următoarea proprietate importantă: șirul care reprezintă translația neterminalului de la începutul fiecărei reguli de producție este concatenarea translațiilor neterminalelor din corpul regulii de producție, în aceeași ordine ca și în regulă. O definiție direcționată de sintaxă cu această proprietate este denumită simplă.

Translatarea expr->t este concatenarea translatărilor lui expr1 și term1, urmată de simbolul +. expr1 și term apar în aceeași ordine atât în ​​corpul regulii de producție, cât și în regula semantică. Nu există simboluri suplimentare înainte sau între translatări. În acest exemplu, singurul simbol suplimentar apare la sfârșit[[21]](#footnote-21).

Parcurgerile de arbori sunt folosite pentru specificarea execuției fragmentelor de cod în cadrul translatării și evaluarea atributelor. O traversare a unui arbore începe de la rădăcină și vizitează fiecare nod al arborelui într-o anumită ordine. Procedura vizitează(N) din Figura 3-7 reprezintă o traversare în adâncime care vizitează copiii unui nod de la stânga la dreapta. Pe parcursul traversării fiecărui nod, este evaluat la final atributul (după ce cu siguranță au fost evaluate atributele copiilor).



Figură 3‑7 - Pseudocodul unei traversări în adâncime a unui arbore de sintaxă și al evaluării atributelor nodurilor

O definiție direcționată de sintaxă nu impune nicio ordine specifică pentru evaluarea atributelor dintr-un arbore de parsare. Orice ordine de evaluare care calculează un atribut după toate celelalte atribute de care depinde este acceptată. Atributele sintetizate pot fi evaluate în timpul oricărei traversări de jos în sus, adică traversarea care evaluează atributele unui nod după ce au fost evaluate atributele copiilor săi[[22]](#footnote-22).

Un element constitutiv poate avea cel mult un atribut asociat. În cazul identificatorilor (numelor de constante, variabile sau funcții) valoarea atributului este un pointer către intrarea din tabela de simboluri aferentă acestuia care permite accesul la mai multe informații[[23]](#footnote-23).

## Tabela de simboluri

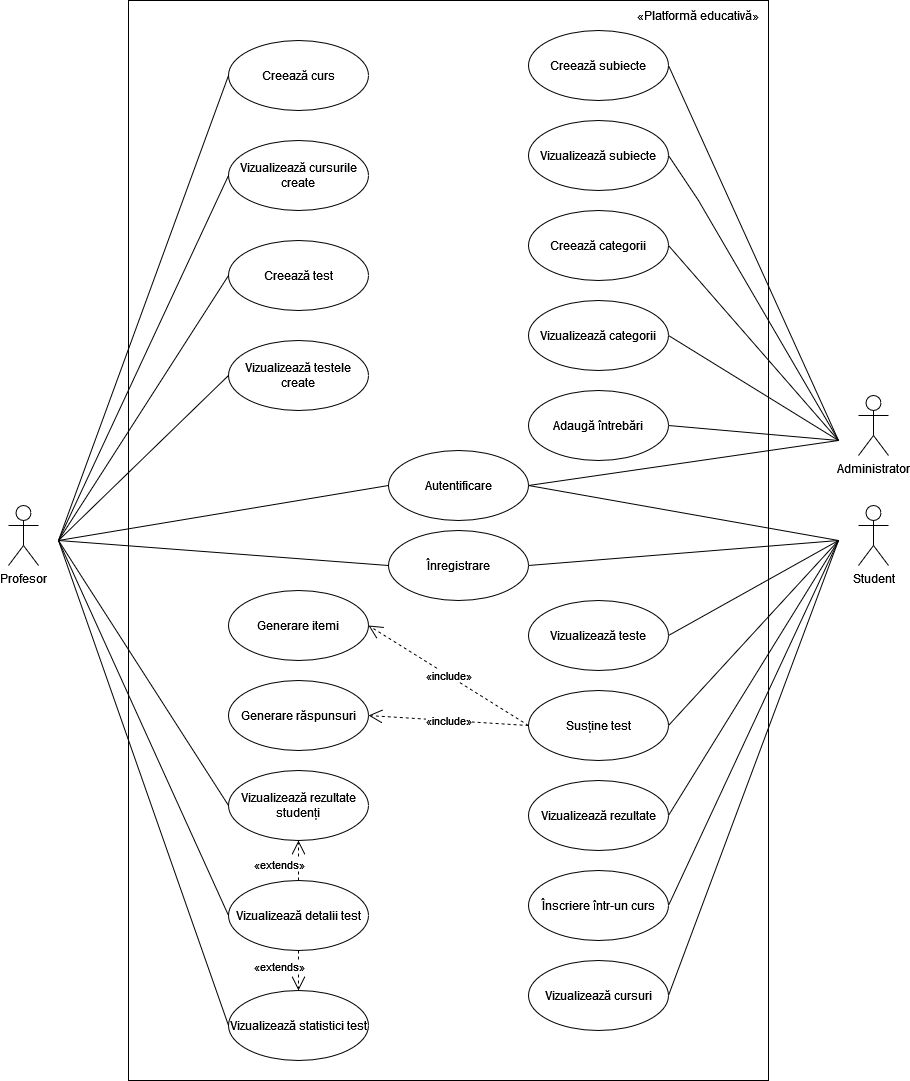
Tabelele de simboluri sunt structuri de date care sunt utilizate de compilatoare pentru a păstra informații despre identificatori (nume de constante, variabile și funcții). Informațiile sunt colectate în mod incremental de fazele de analiză ale unui compilator și utilizate de fazele de sinteză pentru a genera codul țintă. Intrările din tabela de simboluri conțin informații despre identificatori, cum ar fi șirul de caractere asociat(lexemul), tipul lor, adresa lor din memorie și orice alte informații relevante. În cazul procedurilor, sunt stocate informații precum numărul și tipurile argumentelor, metoda de transmitere a fiecărui argument (de exemplu, prin valoare sau prin referință) și tipul returnat[[24]](#footnote-24).

Tabelele de simboluri trebuie de obicei să accepte mai multe declarații ale aceluiași identificator în cadrul unui program (din domenii de valabilitate diferite)[[25]](#footnote-25).

În consecință, rolul unei tabele de simboluri este de a transmite informații de la declarațiile variabilelor către utilizarea acestora. O acțiune semantică pune informații despre identificator în tabela de simboluri, atunci când este analizată declarația acestuia. Ulterior, o acțiune semantică asociată cu o regulă de producție obține informații despre identificator din tabela de simboluri[[26]](#footnote-26).

# Detalii privind implementarea practică

Proiectul are drept obiectiv dezvoltarea unui sistem de evaluare automatizat prin intermediul generării procedurale a întrebărilor și a răspunsurilor aferente acestora. Prin urmare, pentru a îndeplini acest obiectiv, platforma educativă va asigura un serviciu de generare automată a itemilor în timp real pe baza criteriilor selectate în etapa premergătoare testului și a răspunsurilor asociate acestora, precum și o interfață client ce va permite diverse operațiuni în funcție de rolul utilizatorului.

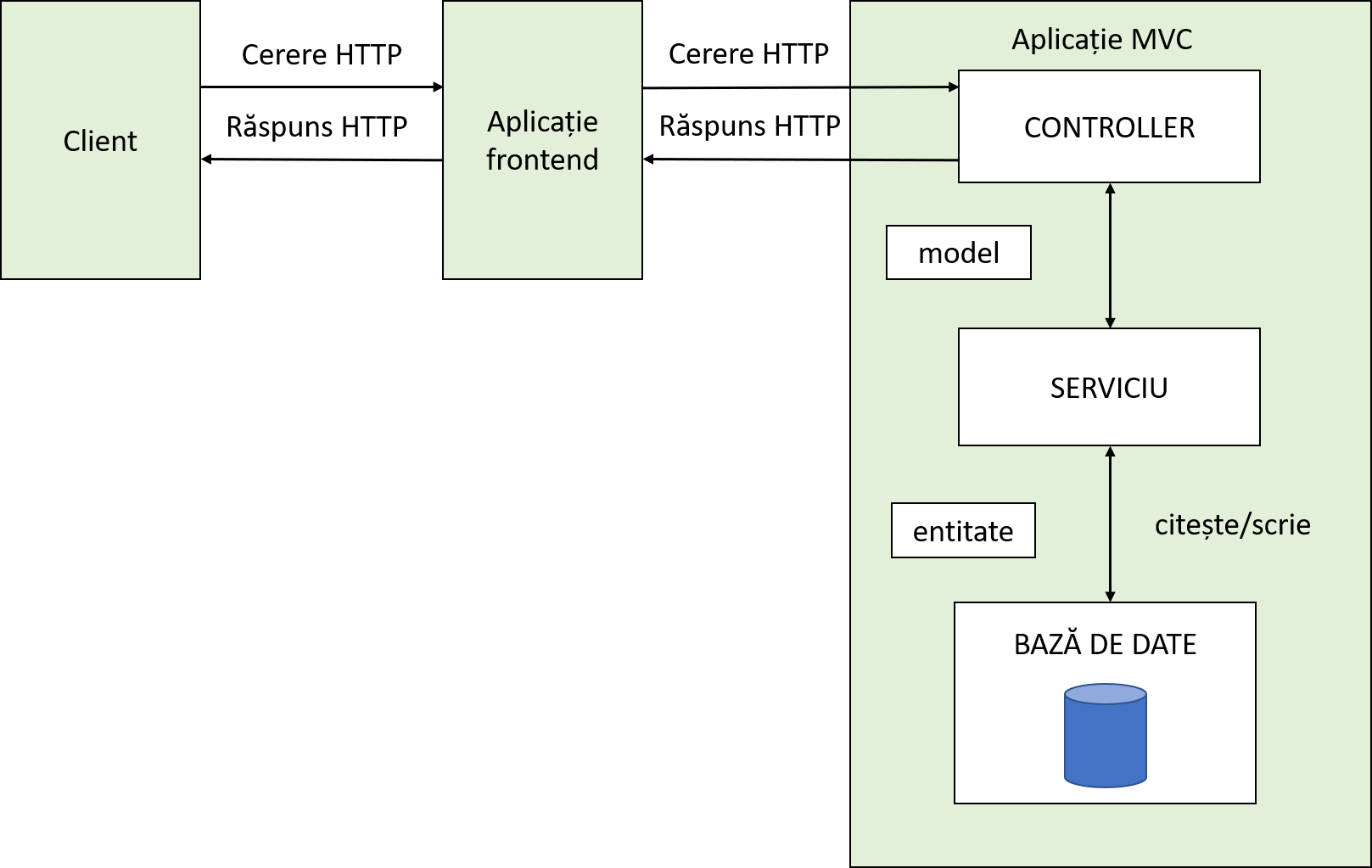


Figură 4‑1 - Diagrama cazurilor de utilizare a platformei educative

Un administrator va putea crea întrebări, în urma adăugării unor subiecte și categorii, pentru o mai bună filtrare a acestora. Un profesor va avea posibilitatea de a crea și vizualiza cursurile și testele create. În urma creării unui test, acesta va putea accesa detaliile asociate precum itemii din componența acestuia, statistici și rezultate disponibile. Un student va putea să se înscrie într-un curs, să vizualizeze cursurile în care este înscris și testele aferente acestora. În momentul în care un test este activ, acesta va avea posibilitatea de a-l susține, iar ulterior de a vizualiza rezultatele testului susținut.

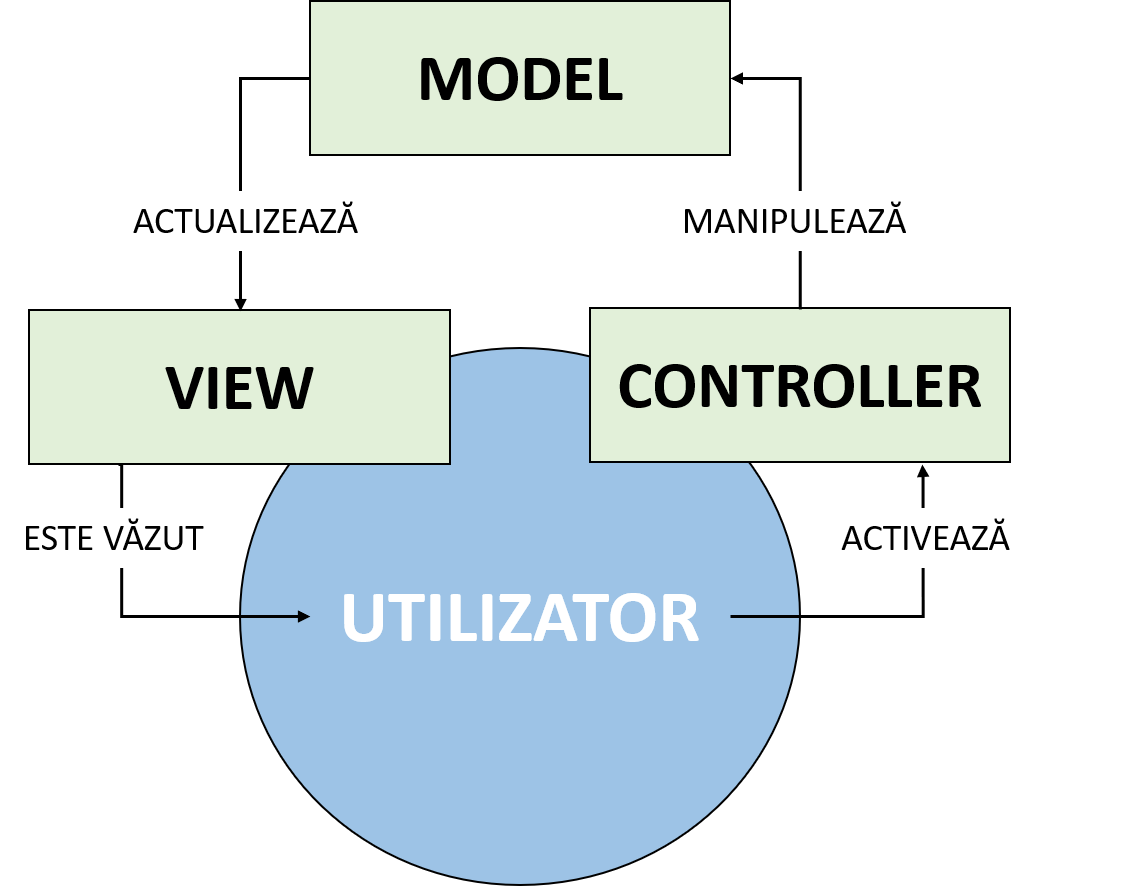
## Arhitectura sistemului

Infrastructura proiectului este una modulară, fiecare parte având roluri bine definite, urmărind modelul propus de arhitectura client-server. Utilizatorul solicită printr-o cerere HTTP o resursă de la server-ul de frontend, care în funcție de necesități, trimite mai departe cereri HTTP către server-ul de backend. Server-ul de backend efectuează procesările complexe și interacționează cu baza de date, iar apoi acesta returnează un răspuns server-ului de frontend, care actualizează componentele interfeței grafice în mod corespunzător și le livrează clientului într-un mod eficient.



Figură 4‑2 - Arhitectura platformei educative de testare

În cadrul proiectului este folosit modelul arhitectural MVC. Rolul acestuia este de a separa logica aplicației de interfața grafică. Astfel, aplicația de backend va fi constituită din două părți principale: model și controller, iar aplicația de frontend va constitui partea de view din MVC. Componenta controller va fi responsabilă cu primirea cererilor și trimiterea răspunsurilor, iar componenta view va descrie interfața grafică. Modelele vor fi folosite pentru a găzdui informația transmisă între celelalte două componente.



Figură 4‑3 - Fluxul acțiunilor în modelul arhitectural MVC

## Tehnologii și API-uri folosite

Pentru implementarea părții de frontend a fost folosită biblioteca Javascript React. Aceasta furnizează un mod facil de creare a unei aplicații de tip SPA (o aplicatie cu o singura pagină) ce interacționează cu browserul web prin rescrierea dinamică, a DOM-ului curent, cu date noi generate fie din client, fie din server. În acest caz, browserul nu este nevoit să încarcă o pagină nouă[[27]](#footnote-27).

Pentru implementarea părții de backend a fost folosit framework-ul ASP.NET Core ce utilizează o implementare a unui server HTTP pentru a gestiona cererile. Serverul furnizează cererile aplicației dezvoltate sub forma unui context HTTP (HttpContext)[[28]](#footnote-28).

O aplicație dezvoltată în React este ideal de integrat cu un backend de tip MVC Web API unde nu există o implementare pentru componenta View[[29]](#footnote-29), iar React poate compensa prin gestionarea părții de interacțiune cu utilizatorul. ASP.NET MVC oferă o modalitate foarte bună de separare a responsabilităților diferitor componente ale aplicației. Această separare este convenabilă pentru dezvoltatori deoarece nu există un mix între codul HTML și logica aplicației[[30]](#footnote-30).

Pentru asigurarea persistenței datelor a fost folosită o bază de date relațională MSSQL, alături de DBMS-ul SQL Server.

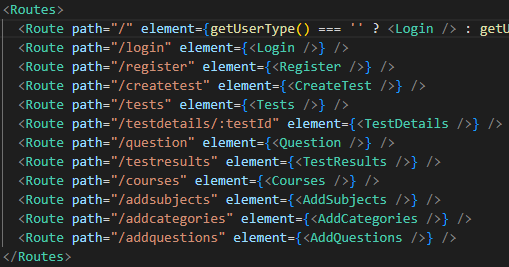
În cadrul aplicației de frontend au mai fost folosite Axios, Bootstrap, Formik, React Router Dom. React Router DOM este un API creat pentru React cu scopul de a ,,simula’’ un browser history. Deoarece aplicația este de tip SPA, browser-ul nu navigheaza prin fișiere html. Axios este o bibliotecă JavaScript folosită pentru a face cereri HTTP din browser. Formik este o bibliotecă Javascript folosită pentru gestionarea facilă a formularelor ce necesită o validare complexă. Bootstrap este un framework CSS utilizat pentru dezvoltarea de aplicații web responsive.

În cadrul aplicației de backend au mai fost folosite BCrypt.Net și Entity Framework Core. BCrypt.Net este o bibliotecă utilizată pentru aplicarea unor funcții de hash asupra parolelor utilizatorilor. Entity Framework Core este un ORM ce oferă dezvoltatorilor un mecanism automat pentru memorarea și accesarea datelor dintr-o bază de date.

De asemenea, au fost furnizate ca dependințe utilitarul clang-format pentru formatarea codului generat și compilatorul C++, g++, folosit prin intermediul mediului de rulare furnizat de Mingw-w64 care asigură funcționalitatea acestuia pe sistemul Windows.

## Implementarea părții de frontend

Comanda create-react-app[[31]](#footnote-31) creează fișierele și instalează modulele necesare pentru a crea o aplicație minimală cu configurările standard necesare, scurtând astfel timpul de dezvoltare. Browser-ul primește de la server-ul de frontend fișierul index.html ce conține un div cu id-ul ,,root’’. Întreaga aplicație va fi randată în interiorul acestui div prin intermediul unei funcții din fișierul index.js asociat. Componenta App conține mai multe rute, care accesate, vor declanșa randarea componentelor aferente[[32]](#footnote-32), a căror interfață poate fi vizualizată în Anexe.



Figură 4‑4 - Componentele create pentru interfața platformei educative

Un avantaj al bibliotecii React este modul de gestionare al navigării între pagini. Prin utilizarea acestei librării este prevenită reîncărcarea totală a unei pagini, lucru care nu este necesar datorită faptului că antetul și finalul componentelor, librăriile JS și CSS nu s-au schimbat. Prin urmare, nu este necesar ca acestea să fie din nou trimise la nivelul clientului.

Rolul principal al unei componente este de a returna codul HTML asociat unei pagini. Celelalte elemente prezente în cadrul unei componente fie gestionează starea acesteia fie transformă anumite date în cod HTML[[33]](#footnote-33).

Un Hook este o funcție specială care permite accesul la niște funcționalități suplimentare oferite de React. Pentru a gestiona starea din interiorul unei componente, a fost folosit hook-ul useState, care returnează două elemente: un obiect căruia îi va fi atribuită o valoare și o funcție de setare a valorii obiectului respectiv. Setarea unui obiect de stare declanșează o rerandare a paginii. De exemplu, tiparul unei întrebări este afișat până în momentul primirii răspunsului de la server, pentru ca ulterior pagina să fie actualizată în mod corespunzător cu întrebarea generată. O altă funcție de tip React Hooks denumită useEffect a fost folosită pentru a executa un ,,efect’’ atât la prima randare a unei componente cât și de fiecare dată când este rerandată. De exemplu, pentru ca un administrator să poată vizualiza categoriile de întrebări create este necesară obținerea datelor de la server. Va fi efectuat un apel către API la prima randare a paginii pentru a fi populată cu intrările corespunzătoare, iar în momentul în care este adaugată o nouă categorie, va fi declanșată o rerandare și prin urmare, un nou apel către API cu scopul de a actualiza pagina.

## Implementarea părții de backend

### Crearea API-ului

Controllerele reprezintă punctul de intrare în partea de server din aplicația client. Acestea definesc rutele (endpoint-urile) pentru API-ul web și gestionează cererile HTTP asociate, orchestrând toate serviciile din backend. Metodele asociate rutelor sunt securizate cu un atribut de autorizare [Authorized]. Această abordare a fost utilizată pentru asigurarea implicită a metodelor și rutelor definite pe parcursul proiectului. Orice metodă nou adaugată este sigură, dacă nu este facută publică în mod explicit prin atributul [AllowAnonymous]. O rută este un tipar URL care este mapat unei acțiuni dintr-un controller MVC.

Tabel 4‑1 - Elemente descriptive ale API-ului platformei de testare

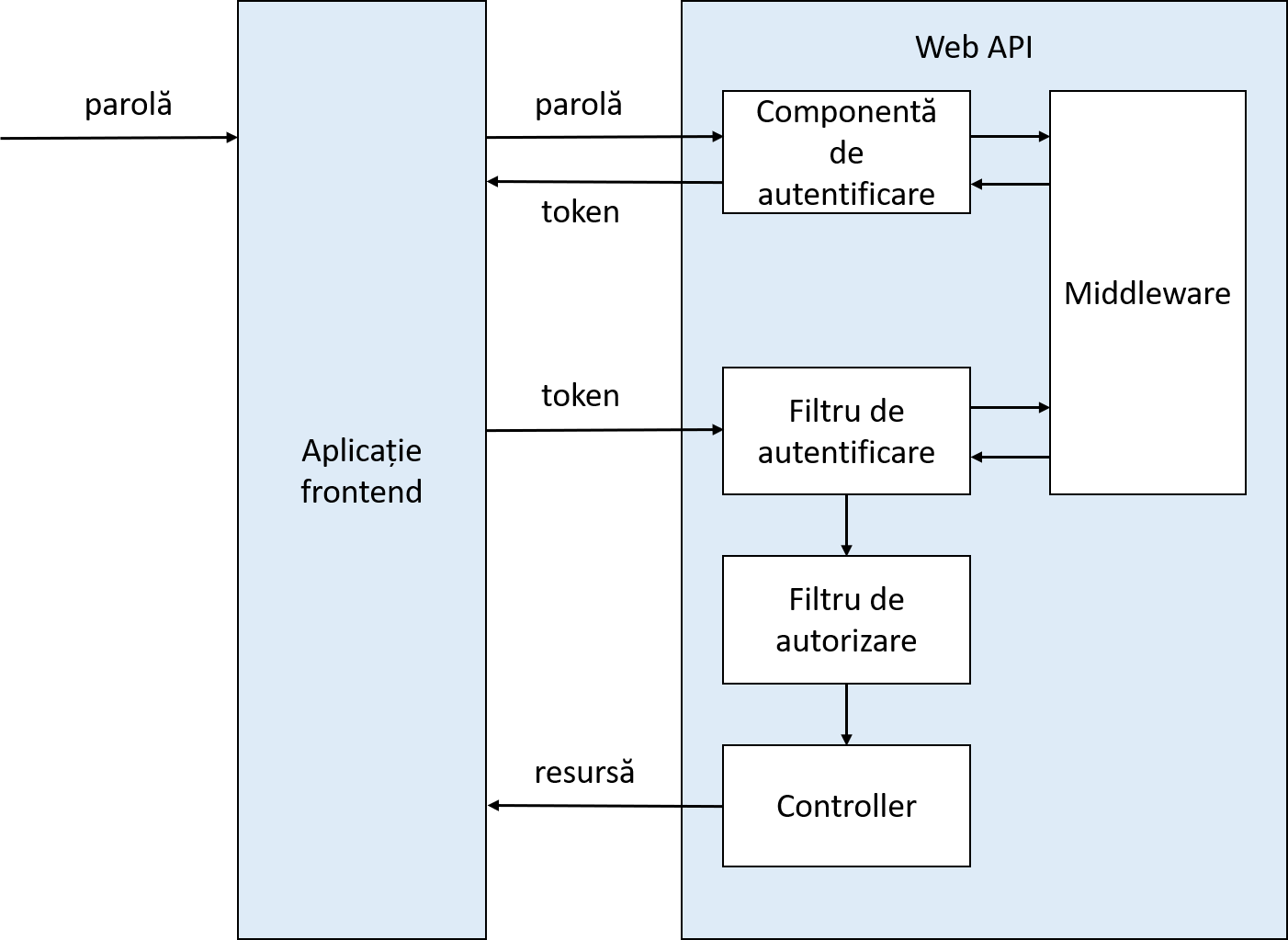
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Utilizatori autorizați | API | | Descriere | Model/Parametrii cerere | Model răspuns |
| administrator | POST | /subjects | Adaugă un subiect | AddSubjectRequest | { message = "The subject has been added" } |
| administrator, profesor | GET | /subjects | Obține subiectele disponibile | - | List<GetSubjectResponse> |
| administrator | POST | /categories | Adaugă o categorie | AddCategoryRequest | { message = "The category has been added" } |
| administrator, profesor | GET | /categories/{subjectId} | Obține categoriile disponibile | subjectId | List<GetCategoryResponse> |
| administrator | POST | /questions | Adaugă o întrebare | AddQuestionRequest | { message = "The question has been added" } |
| administrator, profesor | GET | /questions?subjectId=X&categoryId=X | Obține întrebările asociate unui subiect și unei categorii | subjectId, categoryId | List<GetQuestionResponse> |
| student | GET | /questions/generate?testId=X&username=X&questionTypeId=X | Generează o întrebare de un anumit tip pentru un student care susține un test | testId, username, questionTypeId | GenerateQuestionResponse |
| profesor | POST | /courses | Adaugă un curs | AddCourseRequest | { message = "The course has been created" } |
| student | POST | /courses/enrollment | Înscrie un student într-un curs | AddEnrolledStudentRequest | { message = "The student has been enrolled" } |
| profesor, student | GET | /courses/{username} | Obține cursurile în care un student este înscris sau pe care un profesor le-a creat | username | List<GetCourseResponse>,  List<GetStdCourseResponse> |
| profesor,  student | GET | /courses | Obține toate cursurile disponibile | - | List<GetStdCourseResponse> |
| profesor | POST | /tests | Adaugă un test | AddTestRequest | { message = "The test has been created" } |
| profesor, student | GET | /tests/{username} | Obține testele create de un profesor sau testele asociate cursurilor în care un student este înscris | username | List<GetTestResponse> |
| profesor, student | GET | /tests/test/{testId} | Obține detaliile referitoare la un anumit test precum denumire, dată, oră și rezultate sumare și statistici pentru profesori | testId | GetTestDetailsResponse, GetTestSummaryResponse |
| profesor, student | GET | /results?username=X&testId=X | Obține rezultatele pentru un student care a participat la un anumit test | Username, testid | GetResultResponse |
| student | POST | /results | Adaugă o intrare în tabela de rezultate la începerea unui test de către un student | AddResultReqest | AddResultResponse |
| student | PATCH | /results/details/{resultDetailsId} | Actualizează întrebarea generată cu răspunsul furnizat de student | resultDetailsid, JSONPatchDocument<ResultDetail> | { message = "The result details have been updated" } |
| student | PATCH | /results/{resultId} | Actualizează rezultatul final al studentului cu nota și timpul în care a terminat testul | resultId, JSONPatchDocument<Result> | { message = "The result has been updated" } |
| student | GET | /results/answeredTests/{username} | Obține testele la care a participat un student | username | GetAnsweredTestsResponse |
| - | POST | users/register | Întregistrează un profesor sau un student | RegisterRequest | { message = "Registration successful" } |
| - | POST | users/login | Autentifică un profesor, student sau administrator | LoginRequest | LoginResponse |

### Implementarea autentificării JWT

Pentru a avea acces la funcționalitățile aplicației, utilizatorii trebuie să se autentifice. În urma unei autentificări reușite, este generat un token JWT valid pentru 7 zile ce este semnat folosind o cheie secretă stocată în Secret Manager. Secret Manager este o unealtă care stochează datele confidențiale pe durata dezvoltării unui proiect ASP.NET Core. Secretele sunt stocate într-o locație separată de directorul proiectului, evitând astfel contorizarea lor de către un sistem de versionare[[34]](#footnote-34). Token-ul JWT este returnat aplicației client, care trebuie să îl includă în câmpul Authorization din header-ul cererilor HTTP ulterioare.

Un middleware va verifica dacă există un token în câmpul Authorization din antetul cererii HTTP. Dacă nu există sau dacă acesta nu este valid, atunci niciun utilizator nu este atașat contextului HTTP și cererea poate accesa doar rutele publice. Dacă există, se extrage numele utilizatorului din token și se atașează contextului HTTP.

Procesul de autorizare constă în a verifica dacă există un utilizator atașat cererii curente (contextului HTTP curent). Dacă autorizarea se face cu succes, nu este întreprinsă nicio acțiune și cererea este transmisă mai departe controller-ului. Dacă autorizarea eșuează, este returnat un răspuns cu statusul 401 (neautorizat).



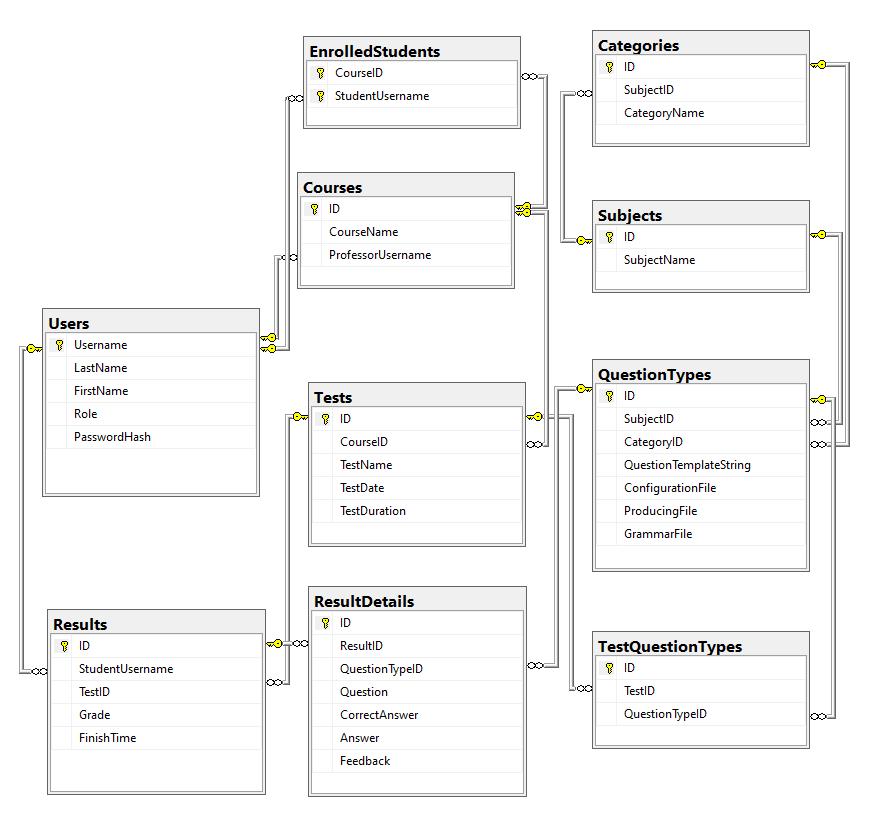
Figură 4‑5 - Procesul de autentificare și autorizare

Rutele restricționate doar utilizatorilor autentificați sunt decorate cu un atribut de autorizare ([Authorized]). Opțional, pot fi specificate doar anumite roluri (de exemplu, profesor, student sau administrator) pentru a restricționa și mai mult accesul. Dacă este specificat un rol, ruta e accesibilă doar utilizatorilor cu acel rol, altfel ruta e accesibilă tuturor utilizatorilor autentificați, indiferent de rol. Astfel, o rută securizată acceptă cereri HTTP și furnizează un răspuns doar dacă antetul cererii conține un câmp de autorizare cu un token JWT valid.

### Interacțiunea cu baza de date

Modelele reprezintă formele pe care le pot lua cererile și răspunsurile pentru metodele controller-ului. Modelele definesc parametrii cererilor care sunt primite sau datele care sunt returnate. Astfel, în momentul primirii unei cereri, conținutul acesteia poate fi deserializat într-un obiect[[35]](#footnote-35). Modelele mai sunt folosite pentru a transmite date între diferite părți ale aplicației (de exemplu, între servicii și controllere).

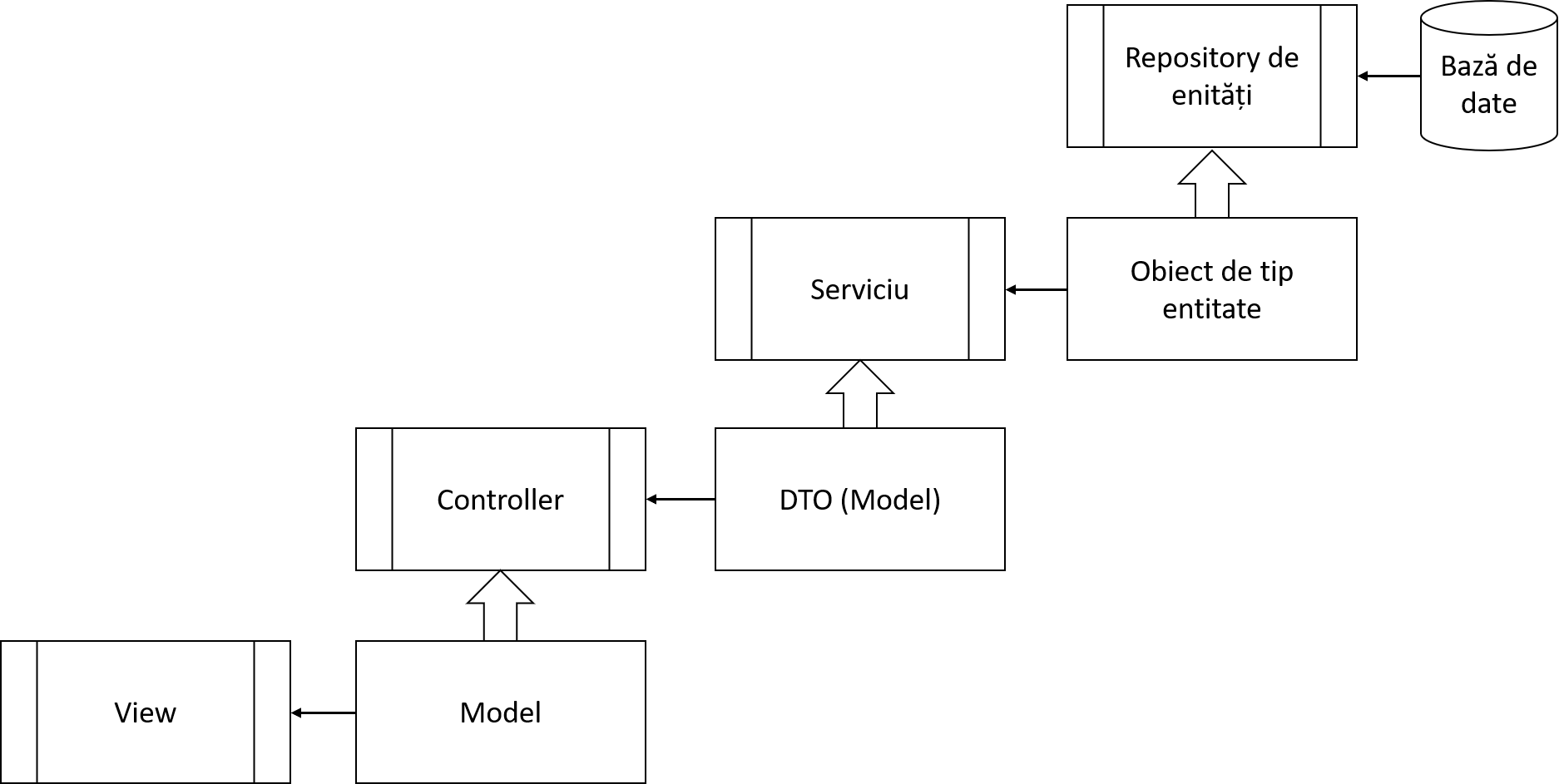
Entitățile modelează datele care sunt stocate în baza de date. Entity Framework Core mapează datele din baza de date relațională cu instanțele unor obiecte, ce sunt utilizate în cadrul aplicației pentru gestionarea datelor și operațiuni CRUD.



Figură 4‑6 - Diagrama bazei de date

Pentru accesarea datelor aplicației este utilizată o clasă asociată contextului bazei de date. Derivă din clasa Entity Framework DbContext și are mai multe proprietăți publice asociate tabelelor, facilitând accesarea și gestionarea datelor în cadrul unei sesiuni[[36]](#footnote-36). Contextul bazei de date este utilizat de servicii pentru a efectua toate operațiunile CRUD.

Tabelele din baza de date relațională sunt mapate sub forma unor entități. Entitățile sunt trimise către servicii, unde sunt procesate (filtrate, combinate, etc.) și convertite în Data Transfer Objects. DTO-urile (modelele) sunt trimise controller-ului, care la rândul lui le trimite către View.



Figură 4‑7 - Fluxul de transmitere al informației

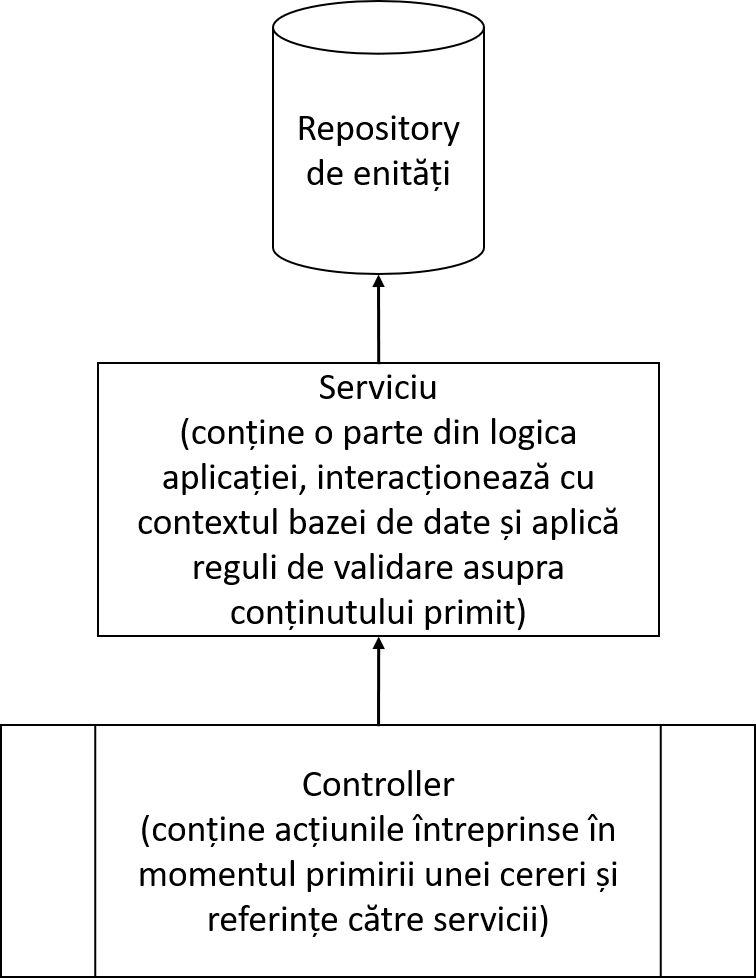
Pe parcursul implementării, a fost necesară luarea unei decizii cu privire la modul de stocare al fișierelor asociate întrebărilor (fișierul de configurare, fișierul producător și fișierul cu gramatica, dacă e cazul). O opțiune este stocarea fișierelor în baza de date. O altă opțiune este stocarea acestora în sistemul de fișiere și referențierea în baza de date. Russell Sears, Catharine van Ingen și Jim Gray (Russell Sears, 2006) au ajuns la concluzia că dacă documentele sunt mai mici de 256KB, stocarea în baza de date este mai eficientă. Dacă documentele au peste un 1MB, stocarea în sistemul de fișiere e mai eficientă. Cum fișierele folosite în cadrul proiectului au mărimi între 0,5 și 6 KB, opțiunea aleasă a fost cea a stocării în cadrul bazei de date.

### Implementarea serviciilor

Una din etapele procesului de implementare a fost definirea locului unde se va afla logica internă a aplicației și implementarea funcționalităților propriu-zise. Modelul poate deveni extins pentru a fi capabil să efectueze o cerere către baza de date și să formateze răspunsul în mod corepunzător, ca să fie returnat View-ului. Însă, cum un model poate fi folosit ca răspuns la o cerere GET sau ca încapsulator al datelor unei cereri POST, acest plus de funcționalitate se poate dovedi redundant. În cazul unei cereri POST nu sunt necesare metodele asociate modelului, ci doar câmpurile postate către server (doar informația).

Dacă logica este pusă în controller, atunci acesta va avea responsabilități pe care nu ar trebui să le aibă, precum validarea datelor de intrare, generarea de modele, formatarea răspunsului, etc. De asemenea, acțiunile întreprinse pentru gestionarea cererilor vor fi greu de citit.

În ambele cazuri, nu sunt respectate principiile POO, clasele devenind din ce în ce mai încărcate, adresând diverse probleme. Astfel, este necesară adaugarea unui nou strat în cadrul arhitecturii aplicației, de care să fie dependent controller-ul. Logica aplicației este la rândul ei, dependentă de datele furnizate de mediul de stocare. Stratul care se ocupă cu logica aplicației poate fi intitulat un serviciu, iar în acest caz ierarhia componentelor, arată ca în figura următoare.



Figură 4‑8 - Poziționarea serviciilor în fluxul de lucru al aplicației

Pentru fiecare cerere, controllerul apelează un serviciu ce efectuează acțiunile necesare, menținându-se complet separat de logica aplicației și de accesul la baza de date. Serviciile conțin logica aplicației, ocupându-se și cu validarea sau accesul la baza de date.

Serviciile dedicate subiectelor, categoriilor și întrebărilor conțin metode ce privesc adăugarea și obținerea acestora de către un administrator. Serviciul de întrebări mai are, adițional, o metodă de generare a întrebării în momentul susținerii unui test de către un student.

Serviciul dedicat testelor conține metode de adăugare a unui test de către un profesor, de obținere a testelor create sau de susținut, în cazul studenților. Acesta mai permite și obținerea detaliilor despre un anumit test precum denumirea acestuia, data, ora, itemii din componență, și în cazul profesorilor, rezultate sumare și statistici referitoare la studenții care au parcurs un anumit test.

Serviciul dedicat rezultatelor conține metode de adăugare a unei intrări în tabela de rezultate, atunci când un student începe un test, de inserare a răspunsurilor la întrebările din test și de actualizare a intrării cu rezultatul final. De asemenea, mai permite și obținerea testelor specifice unui student la care acesta a furnizat răspunsuri pentru a putea fi listate corespunzător în interfață.

Serviciul dedicat utilizatorilor conține metodele necesare pentru accesarea funcționalităților platformei precum cele de înregistrare și autentificare.

Serviciile sunt disponibile în cadrul aplicației prin Dependency Injection. Acestea sunt furnizate componentelor care au nevoie de ele ca parametrii în cadrul constructorilor[[37]](#footnote-37). Constructorul unui controller va primi drept parametrii referințe către serviciile pe care le utilizează. Constructorul unui serviciu utilizează DI pentru a injecta contextul bazei de date folosit pentru orice operație CRUD.

În cadrul fișierului Program.cs este realizată înregistrarea serviciilor pentru dependency injection și a pipeline-ului prin care trec cererile HTTP (middleware). Fiecare serviciu are asociată o singură instanță pe timpul rulării aplicației. Pipeline-ul de gestionare al cererilor este alcătuit din componente denumite middleware care efectuează operații asupra contextului HTTP al acestora și fie invocă următorul middleware fie termină procesarea lor (de exemplu, middleware-ul de autentificare prin JWT) [[38]](#footnote-38).

Configurarea unei aplicații în ASP.NET Core se realizează printr-un furnizor de configurație, care citește datele sub forma cheie-valoare dintr-o sursă precum fișierul appsettings.json.

## Modul de funcționare al generatorului de întrebări

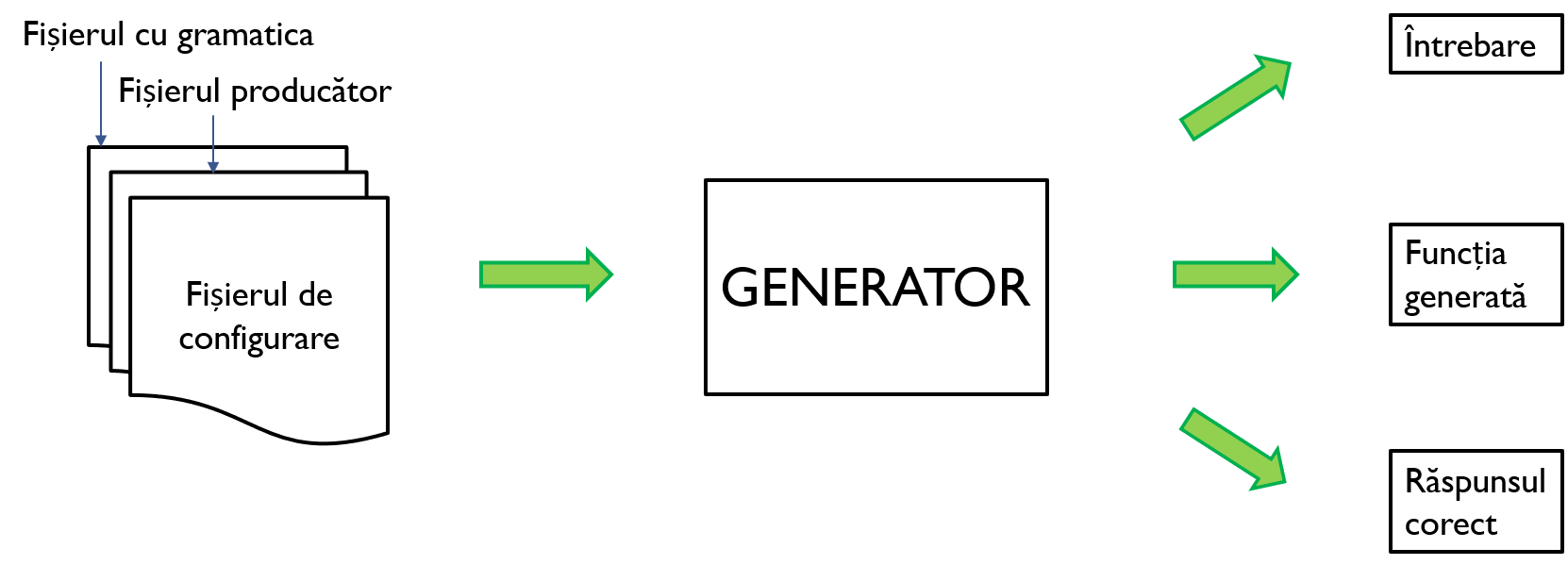
Există o multitudine de tipuri de exerciții care pot fi folosite pentru a evalua cunoștințele de programare ale unui student. În cadrul proiectului, au fost abordate exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi, precum și exercițiile semantice ce constau în înțelegerea fluxului unui program.

Crearea primului tip de exerciții necesită ca generatorul să primească la intrare, un fișier de configurare în format JSON, care definește tiparul întrebării și porțiunile care vor fi înlocuite cu datele produse de programul C++ (fișierul producător), ce constă într-un algoritm precum parcurgerea în preordine/inordine/postordine a unui arbore sau determinarea arborelui parțial de cost minim (Algoritmul lui Kruskal), în urma compilării și rulării.



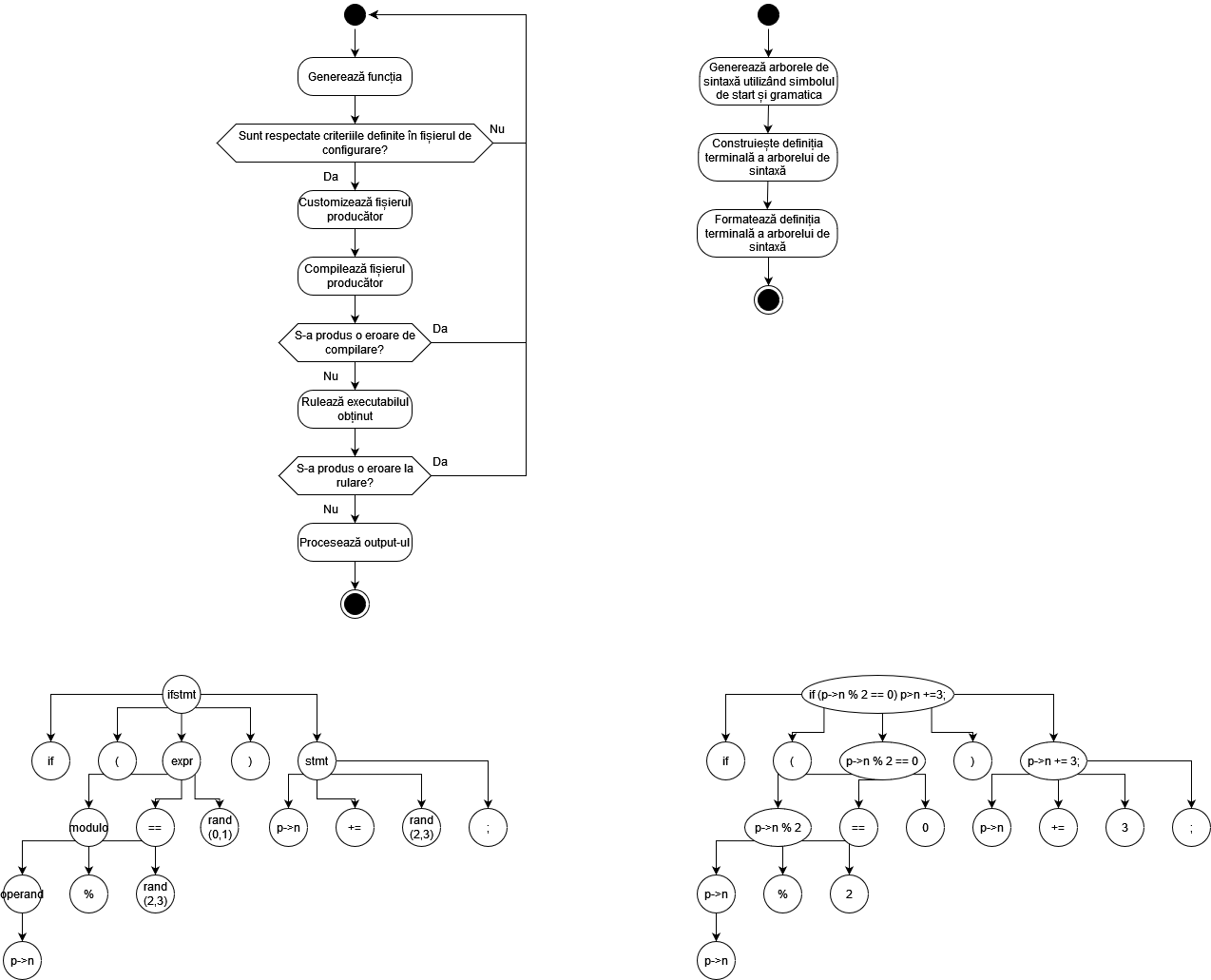
Figură 4‑9 - Schema generală a generatorului pentru exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi

Al doilea tip de exerciții poate fi modelat printr-un triplet alcătuit dintr-o ipoteză, o secvență de cod și un răspuns corect. Generatorul primește ca date de intrare fișierul de configurare în format JSON, fișierul producător ce conține un program C++ care va fi customizat și fișierul cu gramatica ce va fi folosită pentru generarea funcției de customizare a fișierului producător în format JSON. Acesta verifică inițial dacă au fost stabiliți toți parametrii necesari în fișierul de configurare, dacă fișierul producător conține un punct de plecare (metoda main()) și corectitudinea gramaticii.



Figură 4‑10 - Schema generală a generatorului pentru exercițiile semantice

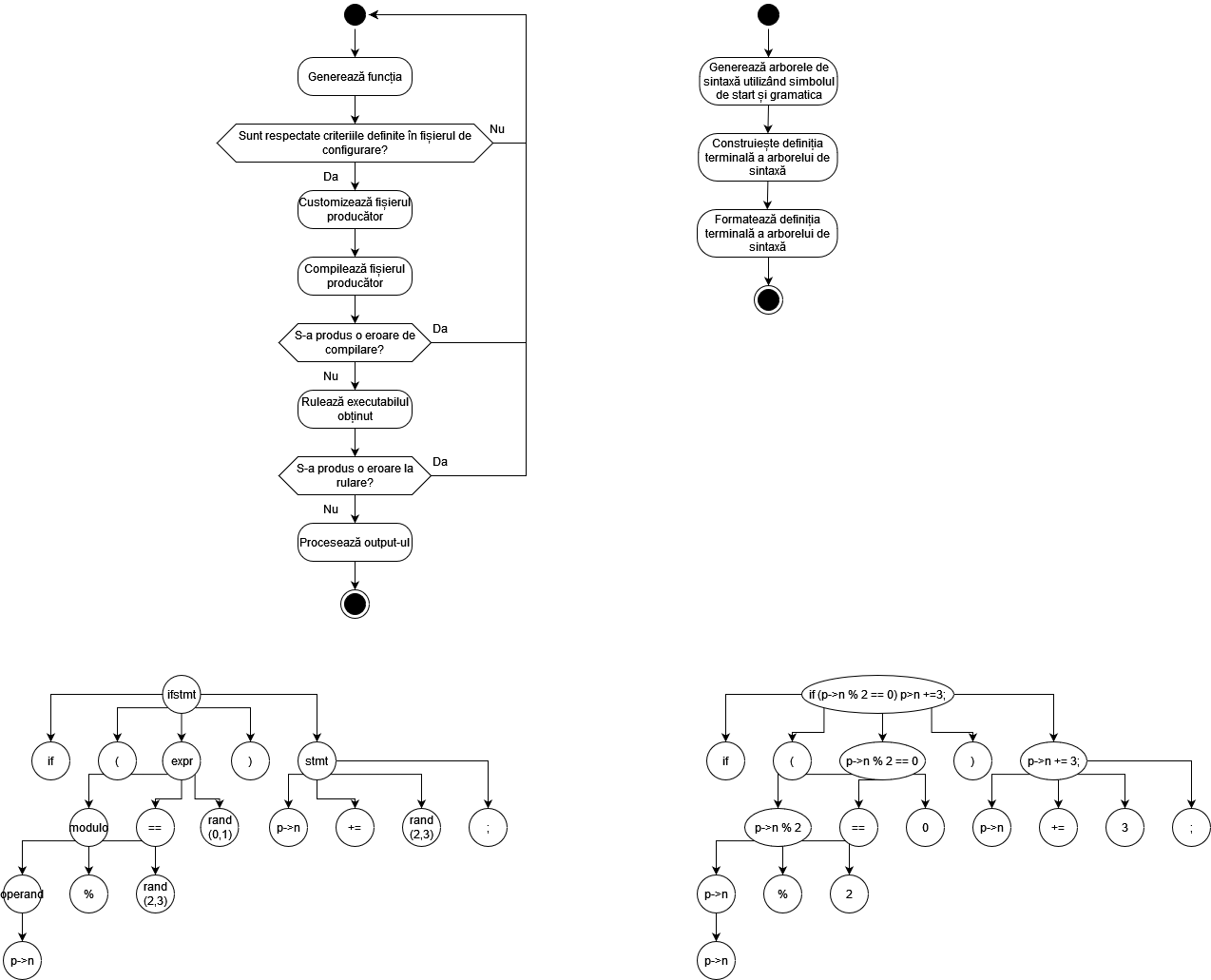
Pentru generarea întregului exercițiu, am definit un pipeline alcătuit din pașii următori: generarea funcției, personalizarea fișierului producător (șablonului), compilarea și execuția acestuia, precum și procesarea output-ului.



Figură 4‑11 - Diagrama de activitate a generatorului

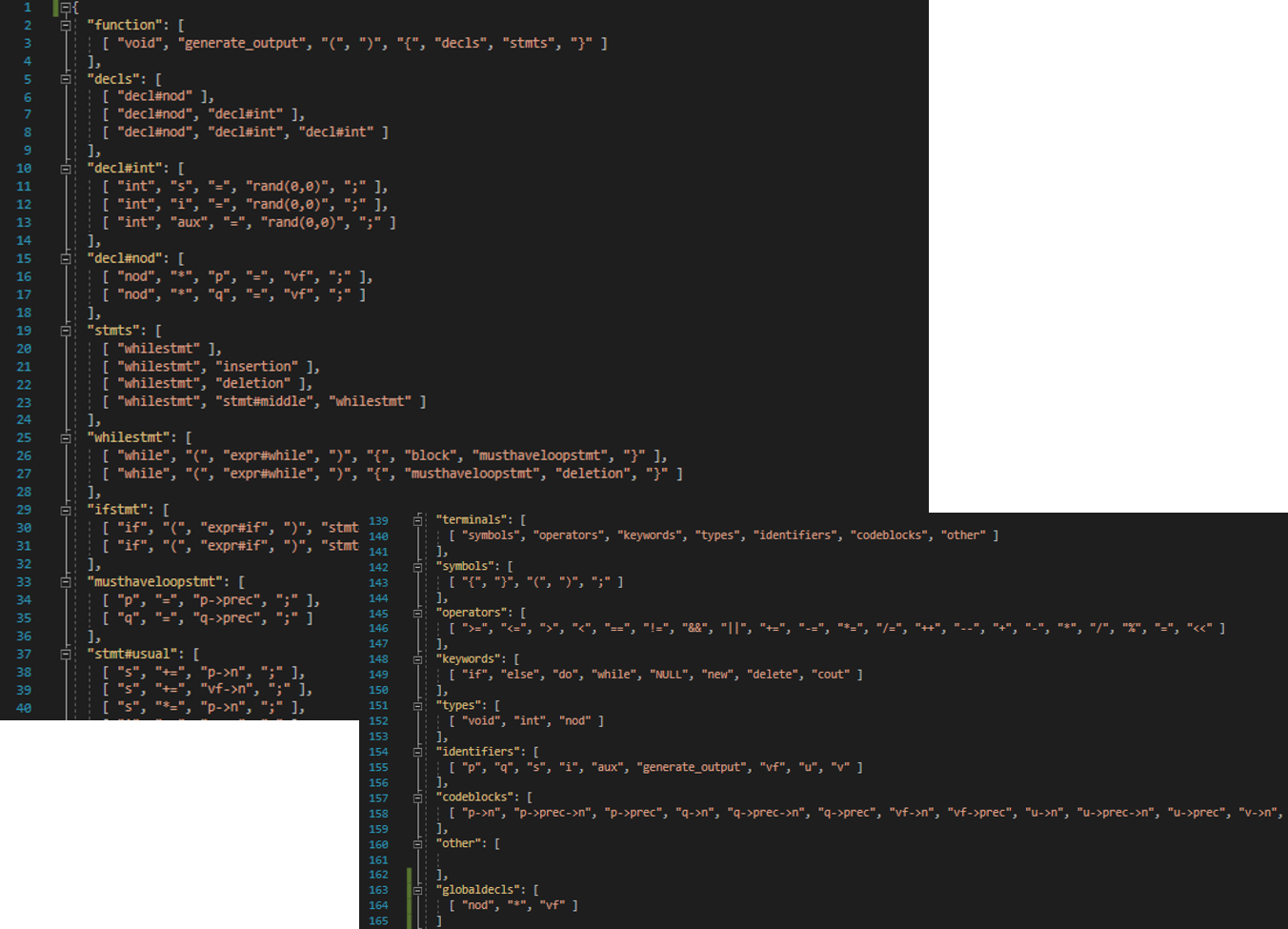
Prin utilizarea acestei abordări, pasul de generare a funcției produce membrul secvenței de cod al tripletului pe care l-am definit anterior. Ipoteza problemei și răspunsul corect sunt generate în etapa de procesare a output-ului, după compilarea și execuția șablonului. Pasul de generare a funcției folosește o gramatică cu ajutorul căreia se generează un arbore de sintaxă folosind o abordare top-down. Pe baza acestui arbore de sintaxă este construită o definiție terminală folosind o abordare de bottom-up.

Generarea funcției se realizează pornind de la simbolul de start definit în fișierul de configurare și utilizând fișierul cu gramatica secvenței de cod. Mai întâi este creat un arbore de sintaxă, pe baza căruia este construită definiția terminală a acestuia. Definiția terminală este, la rândul ei, formatată pentru a putea fi ușor de citit și urmărit de către studenți, în conformitate cu convențiile implicite de scriere a codului utilizate de către utilitarul clangformat. Subsecvent formatării, se verifică dacă secvența de cod generată respectă regulile din fișierul de configurare.



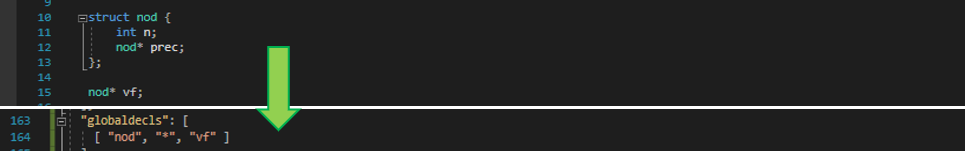
Figură 4‑12 - Diagrama de activitate a pasului de generare a funcției

Structură ierarhică a elementelor ce constituie secvența de cod care se dorește a fi generată este descrisă de o gramatică. Aceasta se mulează pe regulile precise ale limbajului de programare C++ care descriu structura sintactică a programelor corecte (un program este format din funcții, o funcție din declarații și instrucțiuni, o instrucțiune din expresii, etc.). Gramatica este specificată prin enumerarea regulilor de producție aferente simbolurilor. În cadrul proiectului, aceasta mai precizează terminalele (semne de punctuație, operatori, cuvinte cheie, tipuri de date, identificatori sau secvențe de cod) și variabilele globale utilizate în fișierul producător.



Figură 4‑13 - Definiția unei gramatici prin intermediul fișierului în format JSON

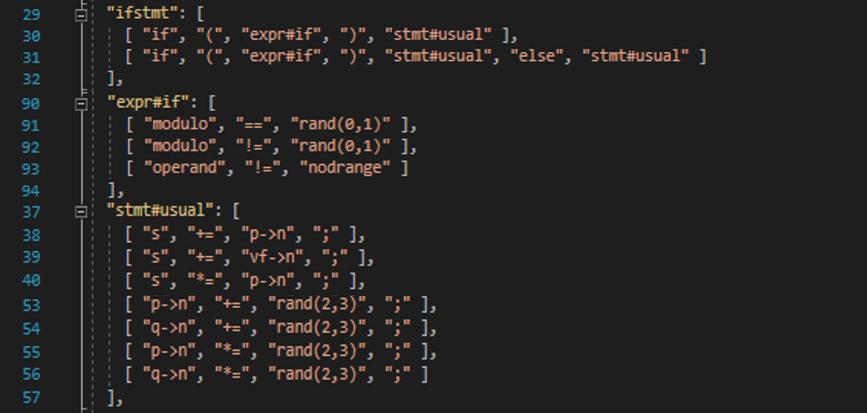
Crearea arborelui de sintaxă are în vedere generarea unui simbol, pe baza șirului de caractere definit ca simbol de start în fișierul de configurare și setarea acestuia drept rădăcină, precum și a structurilor necesare pentru obținerea unui cod calitativ, care să nu genereze erori de compilare (tabela de simboluri, cache-ul de instrucțiuni și stiva cu variabilele folosite în cadrul expresiilor). Premergător generării efective a arborelui, sunt inserate tipurile și identificatorii variabilelor globale din cadrul fișierelor producătoare, care trebuie menționate și în cadrul gramaticii.



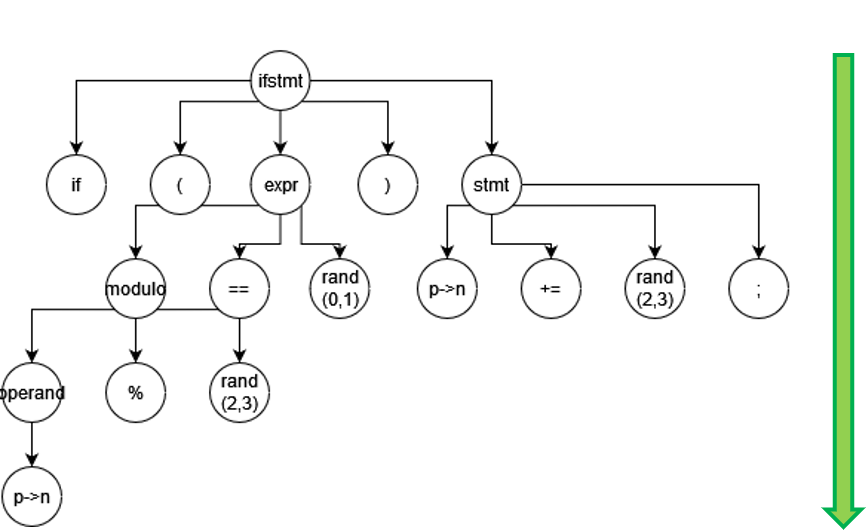
Figură 4‑14 - Transpunerea variabilelor globale din fișierul producător în fișierul cu gramatica

Fiecare nod asociat unui simbol conține o referință către părinte, un identificator (șirul de caractere al simbolului), o listă cu copiii si definiția terminală ce va fi evaluată după generare.

Generarea top-down a arborelui de sintaxă, se face pornind de la rădăcină (simbolul de start). Pentru un simbol este selectată aleator una dintre regulile de producție asociate și vor fi construiți copiii pentru simbolurile din corpul regulii de producție. De la stânga la dreapta, pentru fiecare copil este construit un subarbore. Un simbol nu mai este expandat dacă acesta este un terminal.

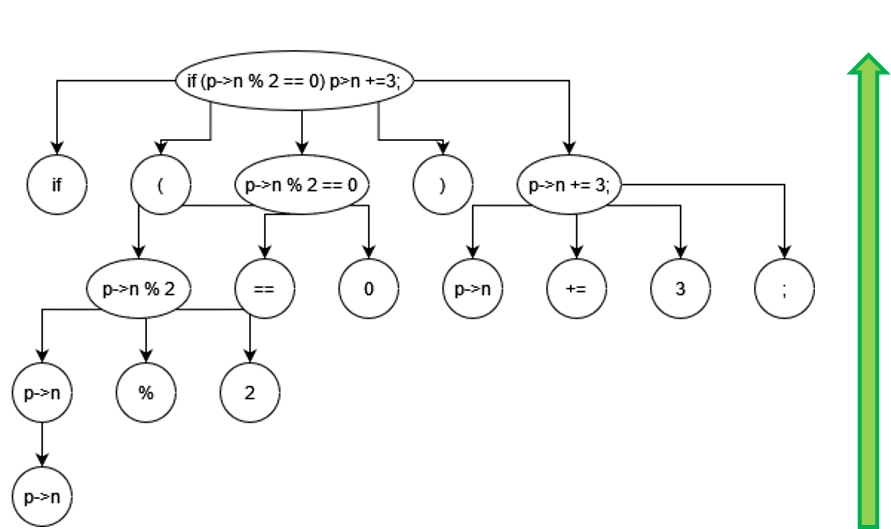


Figură 4‑15 - Definiția gramaticii pentru o instrucțiune condițională și a elementelor constitutive asociate



Figură 4‑16 - Arborele de sintaxă generat pentru o instrucțiune condițională, folosind abordarea top-down

Neterminalele și terminalele au asociate, implicit prin construcția lor, atribute sintetizate (identificatori și definiții terminale) precum și reguli semantice care descriu modul în care sunt întreprinse anumite acțiuni. Aceste reguli sunt executate doar la întâlnirea unor simboluri specifice. Valoarea unui atribut sintetizat a unui nod al arborelui este determinată de valorile atributelor copiilor săi. Atributele sintetizate sunt evaluate în timpul traversării bottom-up a arborelui de sintaxă, în momentul construcției definiției terminale.



Figură 4‑17 - Construcția unei definiții terminale a unei instrucțiuni condiționale, folosind abordarea bottom-up

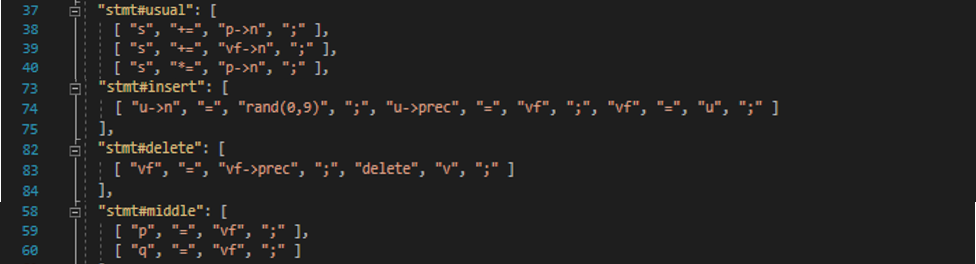
Dacă nodul evaluat este un terminal definiția sa terminală va fi însuși șirul de caractere din care este alcătuit. Dacă nodul este o macrodefiniție precum rand(2,3), definiția terminală va fi obținută prin intermediul unei funcții membru a generatorului, în acest caz returnând un număr aleator din intervalul precizat.

Dacă simbolul începe cu decl, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabila declarată există deja în tabela de simboluri. Dacă generarea este reușită, tipul și identificatorul variabilei vor fi inserate în tabelă.



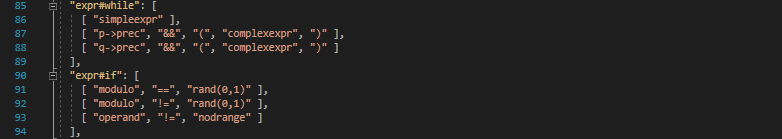
Figură 4‑18 - Regulile de producție asociate unei declarații de variabilă

Dacă simbolul începe cu stmt, va fi generată definiția sa terminală, verificându-se subsecvent dacă instrucțiunea există deja în cache-ul de instrucțiuni. Dacă nu există, vor fi obținute variabilele folosite în cadrul instrucțiunii și se va verifica dacă acestea se află în tabela de simboluri. Dacă se află în tabelă, acestea vor fi marcate ca fiind folosite. Cache-ul de instrucțiuni contine mereu ultimele 2 instrucțiuni generate.



Figură 4‑19 - Regulile de producție asociate unor instrucțiuni

Dacă simbolul începe cu expr, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabilele folosite există deja în tabela de simboluri. În cazul afirmativ, dacă părintele simbolului este o instructiune de tip bucla, acestea vor fi inserate în stiva cu variabilele folosite în expresii și sunt marcate ca fiind folosite în tabela de simboluri.



Figură 4‑20 - Regulile de producție asociate unor expresii

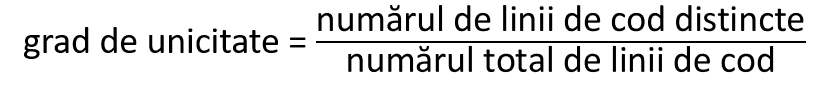
Pornind de la premiza că orice buclă are o instrucțiune care la un moment dat va cauza oprirea acesteia am definit simbolul musthaveloopstmt ca fiind instrucțiunea ce îndeplinește această caracteristică. Dacă simbolul începe cu must, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabilele există în stiva cu variabilele folosite în expresii. Dacă există, va fi efectuată o operațiune de tip pop().



Figură 4‑21 - Regulile de producție asociate unei instrucțiuni iterative din cadrul unei bucle

În oricare dintre aceste cazuri, va fi semnalat dacă variabilele utilizate sunt nedeclarate. La final, sunt eliminate din arbore declarațiile de variabile care au fost nefolosite, verificând dacă în intrările asociate din tabela de simboluri au fost marcate ca fiind folosite.

În urma unei generări reușite, se verifică dacă numărul de linii de cod generate se încadrează între minimul și maximul stabilit în fișierul de configurare. Un utilizator neexperimentat poate crea o gramatică care să permită un grad mai ridicat de recursivitate, și prin urmare, să genereze secvențe de cod de lungimi mai mari decât cele pretabile pentru evaluare. De asemenea, mai este evaluat și gradul de unicitate calculat astfel:



Figură 4‑22 - Definiția gradului de unicitate

Dacă raportul nu depășește gradul de unicitate minim stabilit în fișierul de configurare, va trebui executată o regenerare a funcției.

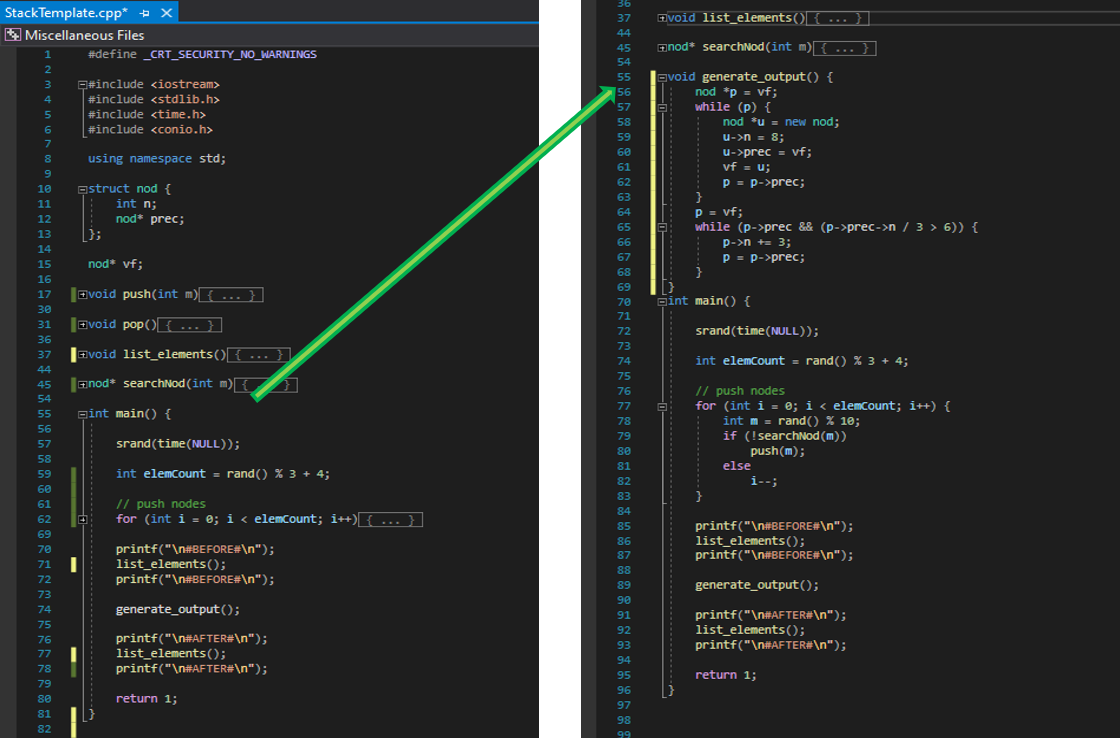
Implementarea structurilor și regulilor semantice menționate anterior, precum și verificările subsecvente au scopul de a genera un cod calitativ, care să nu se repete și să nu producă erori de compilare, scăzând astfel timpul de generare al unei întrebări. Dacă toate criteriile sunt respectate și generarea poate fi considerată reușită, definiția terminală este transmisă restului generatorului pentru procesare ulterioară.

Funcția generată este inserată desupra metodei main() din tiparul desemnat, care va fi compilat și rulat. Fișierul producător (tiparul care va fi customizat cu funcția generată) trebuie:

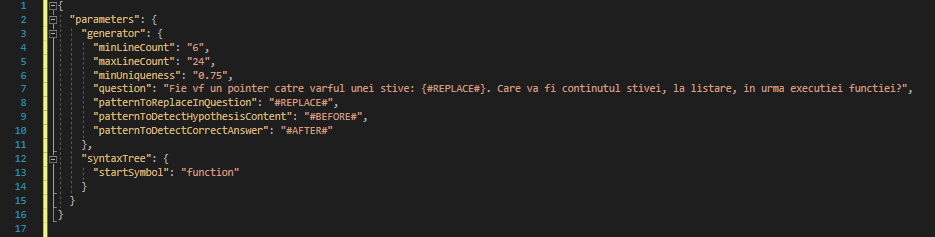
* să conțină variabilele globale ce vor fi utilizate de funcția generată;
* să listeze detaliile ce vor fi inserate în ipoteza din fișierul de configurare;
* să execute funcția generată (inserată înaintea metodei main());
* să listeze răspunsul vizat de întrebarea care folosește șablonul respectiv.

Ambele listări sunt delimitate de tiparele definite în fișierul de configurare.

În cadrul proiectului, pentru întrebările ,,Fie vf un pointer către vârful unei stive: .... Care va fi conținutul stivei, la listare, în urma execuției funcției?’’ și ,,Fie prim un pointer către primul nod al următoarei liste: .... Care va fi conținutul listei, la listare, în urma execuției secvenței următoare?’’, variabilele globale sunt reprezentate de pointerii de tip nod către vârful unei stive, respectiv către primul și ultimul nod ai unei liste. Detaliile ce sunt inserate în ipoteză sunt obținute din listarea conținutului stivei, respectiv listei. Funcția generată alterează conținutul structurilor de date, iar la final prin listarea conținutului curent al structurilor este obținut răspunsul corect.



Figură 4‑23 - Customizarea fișierului producător



Figură 4‑24 - Fișier de configurare

Dacă se produce o eroare de compilare sau de rulare a tiparului personalizat, pipeline-ul va abandona starea curentă și va reporni construcția.

## Descrierea testelor

## Probleme întâmpinate

# Rezultate experimentale

# Concluzii

# Bibliografie

|  |  |
| --- | --- |
| [1] | J. L. B. P. U. S. S. A.-E. Ghader Kurdi, „A Systematic Review of Automatic Question Generation for Educational Purposes,” *International Journal of Artificial Intelligence in Education,* vol. 30, pp. 121-204, 2020. |
| [2] | T. O. Z. S. Danijel Radošević, „Automatic On-line Generation of Student's Exercises in Teaching Programming,” în *Central European Conference on Information and Intelligent Systems*, Varaždin, 2010. |
| [3] | T. M. Akiyoshi Wakatani, „Automatic Generation of Programming Exercises for Learning Programming Languages,” în *IEEE International Conference on Computer and Information Science*, Las Vegas, 2015. |
| [4] | K. S. Thomas James Tiam-Lee, „Procedural Generation of Programming Exercises with Guides Based on the Student’s Emotion,” în *IEEE International Conference on Systems, Man and Cybernetics*, Miyazaki, 2018. |
| [5] | B. K. M. M. Kate Compton, „Tracery: An Author-Focused Generative Text Tool,” în *International Conference on Interactive Digital Storytelling*, Copenhagen, 2015. |
| [6] | „What is automated item generation?,” [Interactiv]. Available: https://assess.com/what-is-automated-item-generation/. [Accesat 8 Iunie 2022]. |
| [7] | „Teste școlare,” [Interactiv]. Available: https://brio.ro/teste-scolare. [Accesat 8 Iunie 2022]. |
| [8] | M. L. R. S. J. U. Alfred Aho, Compilers: Principles, Techniques, and Tools, Boston: Addison Wesley, 2006. |

PAGINA GOALĂ

1. https://en.wikipedia.org/wiki/Procedural\_generation, accesat la 15 mai 2022 [↑](#footnote-ref-1)
2. Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman, Compilers: Principles, Techniques, and Tools, Addison Wesley, Boston, USA, 2006, p. 40 [↑](#footnote-ref-2)
3. Ibidem, p. 109 [↑](#footnote-ref-3)
4. Ibidem, p. 209 [↑](#footnote-ref-4)
5. Ibidem, p. 216 [↑](#footnote-ref-5)
6. Ibidem, p. 97 [↑](#footnote-ref-6)
7. Ibidem, p. 92 [↑](#footnote-ref-7)
8. Ibidem, p. 4 [↑](#footnote-ref-8)
9. Ibidem, p. 191 [↑](#footnote-ref-9)
10. Ibidem, p. 42 [↑](#footnote-ref-10)
11. Ibidem, p. 43 [↑](#footnote-ref-11)
12. Ibidem, p. 44 [↑](#footnote-ref-12)
13. Ibidem, p. 353 [↑](#footnote-ref-13)
14. Ibidem, p. 45 [↑](#footnote-ref-14)
15. Ibidem, p. 61 [↑](#footnote-ref-15)
16. Ibidem, p. 41 [↑](#footnote-ref-16)
17. Ibidem, p. 99 [↑](#footnote-ref-17)
18. Ibidem, p. 54 [↑](#footnote-ref-18)
19. Ibidem, p. 53 [↑](#footnote-ref-19)
20. Ibidem, p. 55 [↑](#footnote-ref-20)
21. Ibidem, p. 56 [↑](#footnote-ref-21)
22. Ibidem, p. 57 [↑](#footnote-ref-22)
23. Ibidem, p. 112 [↑](#footnote-ref-23)
24. Ibidem, p. 4 [↑](#footnote-ref-24)
25. Ibidem, p. 85 [↑](#footnote-ref-25)
26. Ibidem, p. 89 [↑](#footnote-ref-26)
27. https://ocw.cs.pub.ro/courses/pw/laboratoare/03, accesat la 23.04.2022 [↑](#footnote-ref-27)
28. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 25.04.2022 [↑](#footnote-ref-28)
29. https://blog.e-zest.com/setting-up-hybrid-structure-of-mvc-and-reactjs, accesat la 23.04.2022 [↑](#footnote-ref-29)
30. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio, accesat la 25.04.2022 [↑](#footnote-ref-30)
31. https://reactjs.org/docs/create-a-new-react-app.html, accesat la 14.04.2022 [↑](#footnote-ref-31)
32. https://jonhilton.net/understanding-the-asp-net-react-template/, accesat la 24.04.2022 [↑](#footnote-ref-32)
33. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 24.04.2022 [↑](#footnote-ref-33)
34. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-6.0, accesat la 03.05.2022 [↑](#footnote-ref-34)
35. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 25.04.2022 [↑](#footnote-ref-35)
36. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio, accesat la 27.04.2022 [↑](#footnote-ref-36)
37. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio, accesat la 15.05.2022 [↑](#footnote-ref-37)
38. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=windows, accesat la 12.05.2022 [↑](#footnote-ref-38)