**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

****

LUCRARE DE LICENȚĂ

**Titlu**

**propusă de**

***Prenume Nume***

**Sesiunea:** *luna, anul*

**Coordonator științific**

Titlu Prenume Nume

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

Titlu

*Prenume Nume*

**Sesiunea:** *luna, anul*

**Coordonator științific**

*Titlu Prenume Nume*

Avizat,

Îndrumător Lucrare de Licență

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

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

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

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

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

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

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

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

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

DECLARAȚIE DE CONSIMȚĂMÂNT

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

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

Iași, *data*

Absolvent *Prenume Nume*

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

(semnătura în original)

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

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

Încheierea acestui acord este necesară din următoarele motive:

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

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

Iași, *data*

Decan *Prenume Nume* Absolvent *Prenume Nume*

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

(semnătura în original) (semnătura în original)

Cuprins

[Introducere 8](#_Toc517358261)

[Contribuții 9](#_Toc517358262)

[Descrierea problemei 10](#_Toc517358263)

[1.1 Istoric al limbajelor de programare 10](#_Toc517358264)

[1.2 Curba de învățare a limbajelor 15](#_Toc517358265)

[1.2.1 Documentarea 15](#_Toc517358266)

[1.2.2 Instalarea compilatorului sau interpretorului 17](#_Toc517358267)

[1.2.3 Pregătirea mediului de lucru 21](#_Toc517358268)

[1.3 Motivație 23](#_Toc517358269)

[2 Abordări anterioare 24](#_Toc517358270)

[3 Descrierea soluției 25](#_Toc517358271)

[3.1 Arhitectura aplicației 25](#_Toc517358272)

[3.1.1 Front-end 25](#_Toc517358273)

[3.1.2 Back-end 26](#_Toc517358274)

[3.1 Google Cloud 27](#_Toc517358275)

[Concluzii 28](#_Toc517358276)

[4 Bibliografie 29](#_Toc517358277)

# Introducere

De multe ori, când vrem doar să descoperim un limbaj nou, trebuie să trecem prin tot procesul instalăriișsi pregătirii mediului, lucru care de multe ori durează prea mult. De asemenea, există cazuri în care pentru sistemul de operare dorit nu se găsește compilatorul sau interpretorul potrivit sau când ia prea mult timp până reușim să găsim și să instalam toate instrumentele necesare până a ajunge să compilăm o bucată de cod simplă.

În astfel de cazuri, ar fi mult mai utilă o aplicație online prin intermediul căreia să putem scrie și testa cod în diverse limbaje și mai multe versiuni ale acestora. Astfel, nu mai este nevoie de research asupra compilatorului sau interpretorului potrivit sau modului în care putem testa codul. În cazul în care pe calculatorul personal nu mai este destul de mult spațiu pentru a instala un IDE corespunzator, faptul că folosim un compilator online ne scapă de această problemă. Compilând codul în aplicația online, calculatorul personal rămâne liber pentru alte procesări intensive sau pentru aplicațiile folosite de zi cu zi.

Există multe situații în care un programator își dorește doar să verifice și să testeze codul sau doar să învețe un limbaj nou de programare și vrea să descopere care este cel mai potrivit pentru el. În astfel de cazuri, nimeni nu vrea să își ocupe timpul și spațiul din calculatorul personal căutând programe necesare pentru a putea rula un simplu HelloWorld. Soluția cea mai bună în această situație este folosirea unui compilator online, unde toate etapele de pregătire sunt sărite, și tot ce rămâne este scrierea codului după o mică documentare.

Această lucrare compară folosirea unui compilator local cu unul online atunci când un programator se află într-una din situațiile menționate anterior. În descrierea problemei, se prezintă un scurt istoric asupra limbajelor de programare, procesul prin care un programator trece pentru a executa cod scris într-un anumit limbaj și un studiu asupra timpului pierdut în cazul în care vrem să instalăm toate instrumentele necesare unui limbaj de programare. În continuare, se oferă câteva exemple de aplicații asemănătoare și se face o comparație între ele și cea prezentată. La sfârșit, se descrie soluția pentru problema apărută și se oferă o concluzie.

Lucrarea vine împreună cu o aplicație online scrisă în principal în Node.js și AngularJS. Alte tehnologii folosite în dezvoltarea aplicației sunt: HTML, CSS, Bootstrap, Docker, Google Cloud.

Interfața poate fi accesată la momentul scrierii lucrării la adresa <http://35.230.85.239>, ea fiind deployată într-o mașină virtuală din Google Cloud.

# Contribuții

În dezvoltarea părții practice a aplicației, am creat o interfață web de unde utilizatorii pot scrie și testa cod în diverse limbaje. Interfața online are aspectul unui IDE, conținând un editor de cod și butoane pentru personalizarea editorului sau schimbarea dimensiunii textului dar și pentru execuția codului scris. Aceasta permite uploadarea fișierelor din calculatorul personal dar și crearea unor fișiere noi. De asemenea, în mod implicit există un director care conține câte un HelloWorld pentru fiecare limbaj. Limbajele valabile în aplicație sunt: C, C++, C#, Python2, Python3, Java.

Fișierele și conținuturile lor sunt sincronizate la un interval de timp astfel încât, în cazul în care pagina este închisă, la redeschiderea ei starea nu va fi pierdută, conținutul păstrându-se la fel ca înainte. Acest lucru este făcut prin intermediul unui cookie care este setat în browser și care pe partea de server este stocat în baza de date cu toate informațiile legate de fișierele modificate de către utilizator. La redeschiderea paginii, se verifică dacă cookie-ul există și se caută în baza de date sesiunea cu cookie-ul corespunzător, de unde se preiau fișierele.

Aplicația este formată din front-end și back-end. Pentru partea de front-end am folosit HTML, CSS, Javascript, AngularJS și Bootstrap. Pentru back-end, tehnologiile folosite sunt Node.js, ExpressJS, MongoDB și Docker. Întreaga aplicație este urcată în Google Cloud, de unde am creat o mașină virtuală pe care am pornit serverul.

Pentru partea de execuție a codului am folosit Docker, astfel încât am creat inițial câte o imagine de Ubuntu pentru fiecare limbaj valabil în aplicație. De fiecare dată când utilizatorul rulează codul, pe server se creează un nou container bazat pe imaginea de Docker corespunzătoare, se execută codul în interiorul container-ului și se extrage rezultatul, trimițându-l după în interfață pentru a-l vedea clientul. După ce rezultatul este primit, container-ul de Docker este șters de pe server.

La partea de documentație a proiectului, am studiat istoricul limbajelor de programare și am analizat timpul pe care îl consumă un programator atunci când începe să folosească un limbaj de programare pentru prima dată, incluzând documentația, instalarea compilatorului sau a interpretorului sau a unui IDE potrivit.

# Descrierea problemei

## Istoric al limbajelor de programare

O dată cu inventarea mașinii diferențiale de către Charles Babbage în 1882, oamenii de știință au realizat că este nevoie de un mijloc de instruire a computerelor pentru a îndeplini o sarcină anume. Acest mijloc este cunoscut acum ca limbaj de programare și reprezintă o serie de pași ce trebuie executați pentru a realiza un anumit obiectiv. (Ferguson, 2000)

Prima formă a limbajelor de programare a fost de fapt mișcarea fizică astfel încât pentru a executa sarcini, trebuiau schimbate de mână părți din mecanismul mașinii diferențiale a lui Babbage. Ulterior, mișcarea fizică a fost înlocuită de semnale electrice atunci când guvernul american a construit ENIAC[[1]](#footnote-1) în 1942. Acesta a urmat multe din principiile maținii lui Babbage și de aceea putea fi programat doar prin presetarea comutatoarelor și reîncadrarea întregului sistem pentru fiecare nou program sau calcul.

În 1945, în timp ce lucra la Institutul de Studii Avansate, John Von Neumann a dezvoltat două concepte importante care au afectat în mod direct calea limbajelor de programare. Primul a fost “tehnica programelor partajate” ce se referea la faptul că hardware-ul calculatorului trebuie să fie simplu și nu trebuie să fie conectat manual pentru fiecare program. În schimb, ar trebui folosite instrucțiuni complexe pentru a controla hardware-ul simplu, permițând reprogramarea acestuia mult mai rapid.

Al doilea concept a fost extrem de important pentru dezvoltarea limbajelor de programare. Von Neuman a numit-o “transfer de control conditionat”. Această idee a dat naștere noțiunii de subrutine sau mici blocuri de cod ce ar putea fi executate în orice ordine, în schimbul unui set de pași ordonați cronologic. Cea de-a doua parte a ideii a afirmat că programul ar trebui să fie capabil să se încadreze pe baza unor afirmații logice, cum ar fi IF THEN, și cu buclă, cum ar fi o instrucțiune FOR. Conceptul a dat naștere ideii de biblioteci care sunt blocuri de cod ce pot fi refolosite din nou.

În 1949, la câțiva ani după lucrarea lui Von Neuman, a apărut limbajul Short Code. Acesta a fost primul limbaj de calcul pentru dispozitivele electronice și a cerut programatorului să iși schimbe declarațiile în 0 și 1 de mână. În 1951, Grace Hopper a scris primul compilator A-0. Un compilator este un program care transformă declarațiile limbajului în 0 și 1 pentru a face computerul să ințeleagă. Acest lucru a dus la o programare mai rapidă, deoarece programatorul nu mai trebuia să facă lucrul manual.

În continuare, vor fi prezentate cele mai importante limbaje de programare în ordinea în care au apărut.

* Anii 1950
* Autocode (1952) - Această familie de sisteme de codare simplificată a fost creată specific pentru folosirea calculatoarelor digitale din Manchester, Cambridge și Londra. Fiind considerat de mulți ca primul limbaj de programare compilat, Autocode a fost dezvoltat de Alick Glennie cu scopul de a fi ușor de înțeles și de nivel înalt[[2]](#footnote-2).
* Fortran (1957) – Acesta este un limbaj de programare de uz general, imperativ conceput în mod special pentru calcul numeric și calcul științific. Folosit de mai mult de jumătate de secol, Fortran a fost dezvoltat de IBM pentru aplicații inginerești și de știință. Limbajul a ajuns să domine acest domeniu de programare repede, în aplicații precum predicția numerică a vremii, chimie computațională, analiza elementului finit, cristalografie, etc.
* Lisp (1958) – List Processor este o familie de limbaje de programare de nivel înalt creată pentru notații matematice și influențată de notația de calcul lambda a matematicianului Alonzo Church. Lisp a devenit rapid limbajul preferat pentru inteligență artificială pentru că a adus subiecte noi în știința computațională precum structuri de arbori, condiționale, recursivitate, gestionarea automată a memoriei, tipărirea dinamică, funcții de ordin superior, etc.
* Algol 58 (1958) – Numit inițial IAL (International Algebraic Language), este primul limbaj din familia ALGOL și avea ca scopuri principale oferirea unui mijloc de comunicare între oameni a metodelor numerice și a altor proceduri și oferirea unui mijloc de realizare a unui proces declarat pe o varietate de mașini. Algo 58 este limbajul de programare care a introdus conceptul de blocuri de cod și perechea *begin* – *end* pentru a le delimita.
* Cobol (1959) – Common Business-Oriented Language este un limbaj de nivel înalt proiectat în primul rând pentru afaceri, finanțe și administrare a companiilor și guvernului. Acesta a fost creat de Departamentul Apărării din SUA din dorința de a avea un limbaj portabil pentru procesarea informației. Însă, din cauza scăderii din popularitate și retragerea programatorilor cu experiență în acest limbaj, marea majoritate a programelor scrise în Cobol au fost rescrise în alte limbaje moderne.
* Anii 1960
* Algol 60 (1960) – Al doilea limbaj din familia ALGOL, Algorithmic Language 1960 a implementat pentru prima dată definiții de funcții imbricate cu un domeniu de aplicare lexical. Acesta a fost primul limbaj de programare structurat care să conțină blocuri a căror domeniu a definit domeniul variabilelor. Astfel, variabilele declarate în bloc sunt accesibile numai pentru instrucțiunile de cod din acel bloc. Acest concept a devenit unul standard pentru limbajele de programare ce au urmat, el fiind implementat în marea majoritate, excepție făcând BASIC.
* BASIC (1964) – Acronim de la Beginner’s All-purpose Symbolic Instruction Code, BASIC este un limbaj de nivel înalt cu scop general proiectat pentru a fi cât mai ușor de folosit, chiar și pentru cei care lucrau în alte domenii decât știință sau matematică. Popularitatea sa a explodat în mijlocul anilor '70, odată cu apariția calculatoarelor pentru acasă.
* Algol 68 (1968) – Al treilea și ultimul limbaj din familia ALGOL a fost conceput cu scopul unui domeniu de aplicare mult mai larg și cu o sintaxă și semantică mai riguros definite. Algol 68 a introdus multe elemente noi, incluzând vectori flexibili, paralelism, identificarea operatorului, etc.
* Anii 1970
* Pascal (1970) – Numit în onoarea matematicianului francez Blaise Pascal, acest limbaj de programare a fost dezvoltat de Niklaus Wirth. Pascal a permis programatorilor să iși definească propriile tipuri de date complexe și a făcut mai ușor să construiască structuri de date dinamice și recursive precum liste, arbori și grafuri.
* C (1972) – Unul dintre cele mai utilizate limbaje de programare chiar și în prezent, C este un limbaj de uz general destinat programării structurate. Codul sursă scris în C este unul free-format[[3]](#footnote-3), utilizând semnul *;* (punct si virgula) ca terminație a instrucțiunilor și *{*, *}* (acoladele) pentru a grupa blocurile de instrucțiuni. Multe dintre limbajele ce au urmat după C au fost derivate ale acestuia.
* Smalltalk (1972) – Creat în mare parte pentru educație, limbajul a fost lansat inițial ca Smalltalk-80. Acum, se află în continuă dezvoltare activă și a adunat comunități loiale de utilizatori.
* AWK (1977) – Acesta este un limbaj de programare conceput pentru prelucrarea textului și folosit în mod obișnuit ca instrument de extragere și raportare a datelor. Este o caracteristică standard a majorității sistemelor de operare Unix.
* Anii 1980
* Ada (1980) – Numit după Ada Lovelace care este considerată prima programatoare, acesta este un limbaj de nivel înalt derivat din Pascal ce îmbunătățește siguranța și întreținerea codurilor utilizând compilatorul pentru a găsi erori în favoarea erorilor în timpul rulării.
* C++ (1983) – Acest limbaj de programare a fost conceput în principal pentru programarea sistemului, însă a fost extins pentru a fi utilizat în aplicații dektop, servere și aplicații pentru performantă. A moștenit majoritatea sintaxelor C și are caracteristici de programare imperative, orientate pe obiect și generice.
* Objective-C (1983) - Extensie de la C, Objective-C este un limbaj de uz general și nivel înalt ce adaugă funcționalitatea de transmitere a mesajelor pe baza limbajului Smalltalk.
* Eiffel (1986) – Conceput de Bertrand Meyer cu scopul de a spori fiabilitatea dezvoltării software-ului comercial, limbajul este bazat pe o serie de principii strâns legate de metoda de programare Eiffel.
* ABC (1987) – ABC este un limbaj de programare imperativ, de uz general, destinat să fie folosit în loc de BASIC, Pascal sau AWK. Nu este un limbaj de programare a sistemelor, ci este menit predării și prototipării.
* Perl (1987) – Perl este o familie de limbaje de programare de nivel înalt, cu scop general. Acesta împrumută caracteristici din alte limbaje de programare, cum ar fi C, AWK și sed. Perl a fost inițial dezvoltat de Larry Wall pentru a facilita procesarea rapoartelor. De atunci, a suferit numeroase modificari și revizuiri. Perl 6, care a început ca o reproiectare a lui Perl 5 în 2000, a evoluat în cele din urmă într-un limbaj separat. Ambele limbaje continuă să fie dezvoltate independent de diferite echipe de dezvoltare și împrumută idei una de la cealaltă.
* Anii 1990
* Python (1991) – Filozofia de design a lui Python se concentrează asupra lizibilității. Un succesor al limbajului ABC, sintaxa lui permite programatorilor să exprime concepte în mai puține linii de cod decât este posibil în limbaje precum C++ sau Java.
* Visual Basic – Derivat din BASIC, acest limbaj permite dezvoltarea rapidă a aplicațiilor cu interfață grafică pentru utilizator, accesul la baze de date utilizând obiecte de acces de date, obiecte de date la distanță, obiecte de date ActiveX și crearea controalelor și obiectelor ActiveX.
* Ruby (1993) – Dezvoltat de Yukihiro “Matz” Matsumoto în Japonia, Ruby este un limbaj de programare influențat de Perl, Smalltalk, Eiffel, Ada și Lisp. Acesta acceptă mai multe paradigme de programare, inclusiv funcționale, orientate pe obiecte și imperative. De asemenea, dispune de un sistem de tip dinamic și de gestionare automată a memoriei.
* Java (1995) – Acest limbaj de programare a fost conceput pentru a avea cât mai puține dependențe de implementare, oferindu-i o mare varietate de aplicații. Intenția principală este de a permite utilizatorilor să scrie o singură dată dar să ruleze oriunde, astfel încât Java să poată funcționa pe orice platformă care o acceptă fără a fi nevoie să se recompileze.
* PHP (1995) – PHP este un limbaj de scripting pentru partea de server folosit atât pentru dezvoltarea web cât și pentru programarea generală. Acest limbaj poate fi amestecat cu cod HTML sau utilizat în combinație cu motoarele de template și framework-uri web.
* Javascript (1995) – Creat pentru a extinde funcționalitatea paginilor web, limbajul oferă suport pentru depunerea și validarea formularelor, interactivitate, animație, urmărirea activității utilizatorilor, etc.
* Anii 2000
* C# (2001) – Acest limbaj de programare multi-paradigmă a fost dezvoltat de Microsoft în cadrul .NET. A fost gândit să fie simplu, modern și orientat pe obiecte. Cea mai recentă versiune a lui C# a fost lansată în 2012.
* Visual Basic .NET (2001) – Succesor al limbajului original Visual Basic, acesta este un limbaj de programare de nivel înalt implementat în cadrul .NET. Limbajul utilizează instrucțiuni pentru a specifica acțiunile și este unul dintre cele două limbaje principale care vizează cadrul .NET, împreună cu Visual C#.
* Scala (2004) – Scala este un limbaj de programare cu scop general care oferă suport pentru programarea funcțională și un sistem de tip static puternic. Concepute pentru a fi concise, multe dintre deciziile de proiectare ale acestui limbaj au vizat abordarea criticilor pentru Java.
* 2010 până în prezent
* Swift (2014) - Swift a fost creat de Apple pentru dezvoltarea iOS și OS C. A fost introdus la Conferința Mondială a dezvoltatorilor Apple. Conceput pentru a lucra cu framework-urile Cocoa și Cocoa Touch, Swift este menit să fie mai concis și rezistent la cod eronat. (thussong, 2015)

Limbajele de programare au fost în curs de dezvoltare de ani de zile și vor rămâne așa pentru mulți ani în viitor. Au început cu o listă de pași pentru a face un calculator să îndeplinească o sarcină. Acești pași au ajuns în cele din urmă în software și au început să dobândească caracteristici noi și mai bune. Dacă primele limbaje majore s-au caracterizat prin simplul fapt că acestea au fost destinate unui singur scop, cele de astăzi sunt diferențiate prin modul în care sunt programate, deoarece ele pot fi folosite aproape în orice scop.

## Curba de învățare a limbajelor

Calculatoarele nu înțeleg limbile pe care le vorbesc oamenii în viața de zi cu zi. De fapt, la cel mai scăzut nivel, ele înțeleg doar secvențe de numere care reprezintă coduri operaționale. Pe de altă parte, ar fi foarte dificil pentru oameni să scrie programe folosind aceste coduri operaționale. Din acest motiv au fost inventate limbajele de programare pentru a face mai ușor pentru oameni să scrie programe de calculator.

Limbajele de programare au fost create să fie citite și înțelese de oameni. Codul sursă trebuie să fie tradus în limbajul mașinii, astfel încât computerul să poată executa programul. Modul în care se produce această traducere depinde de limbajul folosit, care poate fi unul compilat sau unul interpretat.

Până a ajunge la execuția unei bucăți de cod scrisă într-un anumit limbaj de programare, este nevoie de o serie de pași importanți. Aceștia sunt:

* Documentarea
* Instalarea compilatorului sau interpretorului potrivit limbajului de programare
* Pregătirea mediului de lucru

### Documentarea

Primul pas în procesul de învățare a unui limbaj de programare este cercetarea. Înainte de scrierea efectivă a codului sursă, este nevoie de informarea atât asupra sintaxei limbajului, a funcțiilor sale și modului în care trebuie folosit, dar și a tool-urilor și setările necesare pentru a executa programul, compilatorul sau interpretorul care trebuie instalat și editorul potrivit care să permită scrierea codului într-un mod cât mai ușor.

Înainte de proiectarea aplicației, este nevoie de încadrarea limbajului într-o anumită categorie care să indice modul în care sursa va fi executată: dacă limbajul este unul compilat sau este unul interpretat.

Ambele tipuri de limbaje au puncte forte și puncte slabe. De regulă, decizia de a utiliza un limbaj interpretat se bazează pe restricțiile de timp pe care le dezvoltă sau pe ușurința modificărilor viitoare ale programului. Atunci când se folosește un limbaj interpretat se face de obicei un compromis pentru că se face un schimb între viteza de dezvoltare și costurile de execuție mai ridicate. Deoarece fiecare linie a unui program interpretat trebuie tradusă de fiecare dată când este executată, există o cheltuială mai mare. Astfel, un limbaj interpretat este, în general, mai potrivit pentru cererile ad hoc decât cererile predefinite. (IBM Corporation)

Assembler, COBOL, C, C++ sunt toate traduse prin rularea codului sursă printr-un compilator. Acest lucru are ca rezultat un cod foarte eficient care poate fi executat de mai multe ori. Cheltuielile pentru traducere sunt efectuate o singură dată, când sursa este compilată. Ulterior, trebuie doar să fie încărcat și executat.

Limbajele interpretate, în contrast, trebuie să fie analizate, interpretate și executate de fiecare dată când programul este rulat, contribuind astfel la creșterea costului de rulare a programului. Din acest motiv, programele interpretate sunt de obicei mai puțin eficiente decât programele compilate.

Un dezavantaj al limbilor compilate, este că un program compilat pe un calculator cu Linux nu poate fi copiat direct pe un calculator cu Windows și executabilul să mearga, ci este nevoie ca programul să fie compilat din nou și pe cealaltă mașină ca să poată funcționa.

Unele limbaje de programare, cum ar fi Java si REXX pot fi interpretate sau compilate.

Există motive pentru utilizarea limbilor care sunt compilate și a motivelor pentru utilizarea limbilor interpretate. Nu există un răspuns simplu referitor la ce limbă este mai bună, pentru că depinde de aplicație. Chiar și în cadrul unei aplicații am putea ajunge să folosim multe limbi diferite. De exemplu, unul dintre punctele forte ale unui limbaj ca CLIST este că este ușor de codat, testat și modificat. Cu toate acestea, nu este foarte eficient. Compromisul este resursele mașinilor pentru timpul programatorului.

Ținând cont de acest lucru, putem vedea că ar fi logică folosirea unui limbaj compilat pentru părțile intensive ale unei aplicații, în timp ce interfețele și părțile mai puțin intense ar putea fi scrise într-un limbaj interpretat. O limbă interpretată ar putea fi, de asemenea, potrivită pentru cererile ad-hoc sau chiar pentru prototiparea unei aplicații.

Una dintre funcțiile unui programator este să cântărească punctele forte și slăbiciunile fiecărei limbi și apoi să decidă care parte a unei aplicații este cel mai bine servită de o anumită limbă.

Pe lângă informarea asupra compilatorului sau interpretorului, este necesară documentarea asupra sintaxei limbajului. În general, pentru a învăța un limbaj de programare nou, oamenii pornesc de la un exemplu simplu care doar afișează un text, cod care în lumea programatorilor este numit “Hello World!”. Un exemplu de astfel de cod sursă este în Fig. 1.

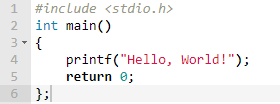


Fig. 1 – Hello World în limbajul C

Cel mai ușor mod de a te familiariza cu un limbaj nou de programare este de a scrie cod singur. Se poate începe cu un tutorial interactiv precum Codecademy[[4]](#footnote-4) unde noțiunile se iau treptat și utilizatorul scrie bucăți de cod care să îndeplinească un anumit scop. LeetCode[[5]](#footnote-5), HackerRank[[6]](#footnote-6) și Project Euler[[7]](#footnote-7) sunt alte platforme create pentru familiarizarea cu diferite limbaje și concepte într-un mod cât mai plăcut și distractiv, prin diferite provocări sau jocuri. De asemenea, există multe site-uri de comunități unde se găsesc răspunsuri la foarte multe întrebări și probleme cu care programatorii se întâlnesc, în orice limbaj ar lucra. Astfel de site-uri sunt StackOverflow[[8]](#footnote-8) sau Code Review[[9]](#footnote-9) unde poți primi recenzii asupra codului tău de la experți în domeniu și nu numai.

Cel mai important însă când vine vorba de învățarea unui limbaj de programare nou este documentația oficială a limbajului, care poate fi găsită atât online cât și downloadată pentru a fi accesată offline. Documentația conține structura și sintaxa limbajului, toate funcțiile construite în mod implicit cu parametrii lor și ce returnează ele.

### Instalarea compilatorului sau interpretorului

Există diferite moduri în care putem executa programe. Compilatoare, interpretoare sau mașini virtuale sunt câteva instrumente pe care le putem folosi pentru a îndeplini această sarcină. Toate aceste instrumente oferă o modalitate de a simula în hardware semantica unui program. Deși aceste tehnologii diferite au același scop, de a executa un program, o fac în moduri foarte diferite. În principiu, orice limbaj de programare poate fi compilat sau interpretat. Cu toate acestea, unele strategii de execuție sunt mai naturale în unele limbaje decât altele. (Introduction to Programming Languages)

Compilatoarele sunt programe de calculator care traduc un limbaj de programare de nivel înalt la un limbaj de programare de nivel scăzut. Produsul unui compilator este un fișier executabil, care este alcătuit din instrucțiuni codificate într-un anumit cod de mașină. Prin urmare, un program executabil este specific unui tip de arhitectură a calculatorului. Compilatoarele concepute pentru limbaje de programare distincte ar putea fi destul de diferite între ele. Cu toate acestea, toate au tendința de a avea macro-arhitectura descrisă în Fig. 2.

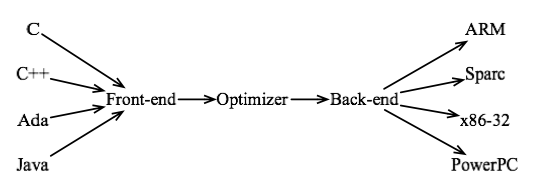


Fig. 2 – Arhitectura unui compilator tipic, format din back-end, front-end și un optimizator de cod

Un compilator are un front-end, modulul care se ocupă de transformarea unui program într-o reprezentare intermediară pe care compilatorul o va procesa în următoarele faze. Unele compilatoare, precum *gcc*, pot parsa mai multe limbaje de intrare diferite. În acest caz, compilatorul are câte un front-end pentru fiecare limbă pe care o poate gestiona.

Un compilator are și un back-end, care generează coduri. Dacă compilatorul poate viza mai multe arhitecturi de calculatoare diferite, atunci va avea un back-end diferit pentru fiecare din ele. În cele din urmă, compilatoarele fac în general o anumită optimizare a codului. Cu alte cuvinte, el încearcă să îmbunătățească programul, având în vedere un anumit criteriu de eficiență, cum ar fi consumul de viteză, spațiu sau energie. În general, optimizatorul nu are voie să modifice semantica programului de intrare.

Principalul avantaj al execuției prin compilare este viteza. Deoarece programul sursă este tradus direct în codul mașinii, acesta va fi mult mai rapid decât dacă ar fi interpretat. Există însă situații în care un program interpretat funcționează mai repede decât codul mașinii echivalent. Principalul dezavantaj al execuției prin compilare este portabilitatea. Un program compilat vizează o arhitectură specifică a calculatorului și nu va putea să ruleze într-un hardware diferit.

Interpretoarele execută programele într-un mod diferit. Ele nu produc cod binar nativ. În schimb, un interpretor convertește un program într-o reprezentare intermediară, de obicei un arbore pe care îl traversează cu un algoritm.

Programul sursă este lipsit de semnificație pentru interpretor în forma sa originală. Astfel, un interpretor, exact ca și un compilator, trebuie să parseze programul sursă. Cu toate acestea, spre deosebire de un compilator, interpretorul nu analizează întregul cod sursă înainte de a-l executa, ci doar acele bucăți care sunt accesibile din fluxul de execuție.

Compilatorul sau interpretorul care va fi instalat trebuie ales în funcție de limbajul de programare și sistemul de operare al dezvoltatorului. Pentru fiecare limbaj există diverse compilatoare sau interpretoare printre care se enumeră (List of compilers):

* pentru Ada:
* *PTC ObjectAda -* pentru Windows, Unix și alte sisteme de operare (compilator)
* *GCC GNAT -* pentru Windows și Unix (compilator)
* *Score Ada -* pentru Windows, Unix și alte sisteme de operare (compilator)
* pentru Algol:
* *ALGOL 60-* pentru DOS și CP/M (compilator)
* *MCP –* pentru MCP (compilator)
* pentru BASIC:
* *BlitzMax -* pentru Windows, Linux, OS X (compilator)
* *Gambas -* pentru Unix (compilator)
* *PureBasic -* pentru Windows, Unix și alte sisteme de operare (compilator)
* *Qbasic-* pentru Windows (interpretor)
* *Chipmunk Basic -* pentru Windows, Unix și alte sisteme de operare (interpretor)
* pentru C:
* *GCC -* pentru Windows (MinGW, Cygwin), Unix, IBM mainframe, AmigaOS, VAX/VMS, RTEMS, DOS (compilator)
* *Clang -* pentru Windows, Unix si alte sisteme de operare (compilator)
* *Aztec C -* pentru CP/M, CP/M-86, Classic Mac OS (compilator)
* *BDS C -* pentru CP/M (compilator)
* *Ch -* pentru Windows, OS X, FreeBSD, Linux, Solaris, HP-UX, AIX, Qnx (compilator)
* *Interactive C -* pentru Windows, Unix, OS X, Linux, IRIX, Solaris, SunOS (compilator)
* *Icc -* pentru toate sistemele de operare (compilator)
* *MPW C -* pentru Classic Mac OS (compilator)
* *Open64 -* pentru Unix (compilator)
* *VBCC -* pentru toate sistemele de operare (compilator)
* *XL C -* pentru AIX, Linux (compilator)
* pentru C++:
* *Turbo C++(tcc) -* pentru DOS (compilator)
* *CINT -* pentru toate sistemele de operare (compilator)
* *GCC -* pentru Windows (MinGW, MSYS2, Cygwin) și alte sisteme de operare (compilator)
* *Intel C++ Compiler (icc) -* pentru Windows și Unix (compilator)
* *Open64 -* pentru Unix (compilator)
* *PGC+ (pgc++) -* pentru Linux, macOS (compilator)
* *C++ Builder (bcc\*) -* pentru Windows (bcc32c, bcc64), iOS (bccios\*), Android (bcca\*) (compilator)
* *CINT -* pentru toate sistemele de operare (compilator)
* *Visual C++ (cl) -* pentru Windows, Unix (compilator)
* *Diab Compiler (dcc) -* pentru Windows, Linux, Solaris, VxWorks (compilator)
* pentru C#:
* *Visual C# -* pentru Windows, iOS (compilator)
* *Visual C# Express –* pentru Windows (compilator)
* *Mono –* pentru toate sistemele de operare (compilator)
* *Portable.NET –* pentru Windows, Unix (compilator)
* *SharpDevelop –* pentru Windows (compilator)
* *Roslyn –* pentru Windows și parțial pentru Unix (compilator)
* pentru COBOL:
* *IBM COBOL -* pentru Windows, AIX, z/OS, z/VM (compilator)
* *GnuCOBOL -* pentru toate sistemele de operare (compilator)
* *MCP -* pentru MCP (compilator)
* pentru Fortran:
* *G95 -* pentru toate sistemele de operare (compilator)
* *GNU Fortran -* pentru toate sistemele de operare (compilator)
* *Open64 -* pentru toate sistemele de operare (compilator)
* *XL Fortran -* pentru Linux (POWER și Cell), AIX, BlueGene (compilator)
* *MCP -* pentru MCP (compilator)
* *Cray -* pentru toate sistemele de operare (compilator)
* pentru Go:
* *Gc -* pentru toate sistemele de operare (compilator)
* *Gccgo -* pentru MinGW, Cygwin, Unix (compilator)
* pentru Java:
* *javac -* pentru toate sistemele de operare (compilator)
* *gcj -* pentru Unix (compilator)
* *ECJ (Eclipse Compiler for Java) -* pentru toate sistemele de operare (compilator)
* pentru Pascal:
* *Delphi -* pentru Windows, Linux, Mac OS, iOs, Android (compilator)
* *GNU Pascal -* pentru toate sistemele de operare (compilator)
* *Turbo Pascal -* pentru alte sisteme de operare în afară de Windows și Unix (compilator)
* *Virtual Pascal -* pentru Windows, Unix, OS/2 (compilator)
* *MCP -* pentru MCP (compilator)
* pentru Python:
* *Cython -* pentru toate sistemele de operare (compilator)
* *Psyco -* pentru toate sistemele d eoperare (compilator)
* *Pypy -* pentru toate sistemele de operare (compilator)
* *Shed Skin -* pentru toate sistemele de operare (compilator)
* *Python -* pentru toate sistemele de operare (interpretor)

### Pregătirea mediului de lucru

Deși pentru a scrie cod poate fi folosit orice editor de text, este indicat să fie folosit un editor special pentru limbajul de programare folosit. Un astfel de editor este mult mai eficient deoarece recunoaște sintaxa limbajului și astfel evidențiază prin diferite culori cuvintele cheie folosite în limbaj, subliniază atunci când apare o eroare de sintaxă și poate da recomandări de cuvinte în timpul tastării atunci când este activată funcția de auto-complete.

Fișierul creat va putea fi rulat folosind comanda specifică compilatorului. De exemplu, pentru limbajul C și fișierul *hello\_world.c* în care se află conținutul din Fig. 1, în cazul în care se va folosi compilatorul GCC, este necesară comanda:

*gcc hello\_world.c*

Rezultatul comenzii va fi:

*Hello, World!*

Pe lângă varianta cu editorul și compilatorul separate, mai este și opțiunea de a folosi un mediu de dezvoltare integrat (IDE) care să le conțină pe ambele și care este mult mai eficient și ușor de folosit.

Un IDE este o aplicație ce facilitează dezvoltarea aplicațiilor. IDE-urile sunt concepute pentru a cuprinde toate sarcinile de programare într-o singură aplicație. Prin urmare, ele oferă o interfață centrală care include toate instrumentele necesare dezvoltatorului, inclusiv următoarele:

* Editor de cod – este un editor de text conceput pentru scrierea și editarea codului sursă. Editoarele de coduri sursă se disting de editoarele de text deoarece sporesc sau simplifică scrierea și editarea codului.
* Compilator – care transformă codul scris într-o limbă umană într-o formă executabilă de către un computer
* Debugger – instrument folosit în timpul testelor pentru a ajuta la depanarea programelor și găsi erori
* Instrumente de automatizare – automatizează sarcini comune ale dezvoltatorilor (WHAT IS AN INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)?)

În plus, unele IDE-uri conțin și:

* Navigator de clase – instrument folosit pentru a examina și referi proprietățile unei ierarhii de clase orientate pe obiecte
* Navigator de obiecte – instrument utilizat pentru a examina obiectele instanțiate într-un program de aplicație în desfășurare
* Diagrama ierarhiei claselor – instrument ce permite programatorului să vizualizeze structura codului de programare orientat pe obiecte.

Scopul general și beneficiul principal al unui mediu integrat de dezvoltare este îmbunătățirea productivității dezvoltatorilor. IDE-urile sporesc productivitatea prin reducerea timpului de instalare, creșterea vitezei sarcinilor de dezvoltare, păstrarea programatorilor la zi cu tot ce apare nou și standardizarea procesului de dezvoltare.

* Setare mai rapidă: fără o interfață IDE, dezvoltatorii ar trebui să-și petreacă timpul pentru a configura mai multe instrumente de dezvoltare. Cu integrarea aplicației într-un IDE, dezvoltatorii au același set de capabilități într-un singur loc, fără a fi nevoie să se schimbe în mod constant instrumentele.
* Task-uri mai rapide: integrarea mai strânsă a tuturor task-urilor de dezvoltare îmbunătățește productivitatea dezvoltatorilor. De exemplu, codul poate fi analizat și sintaxa verificată în timp ce este editat, furnizând feedback instantaneu atunci când sunt introduse erorile de sintaxă.Programatorii nu trebuie să schimbe aplicațiile pentru a îndeplini fiecare task pe rând. În plus, instrumentele și funcțiile IDE ajută dezvoltatorii să organizeze resurse, să prevină greșelile și să facă comenzi rapide.
* Învățare continuă: subiectele de ajutor din IDE sunt actualizate în mod constant, la fel ca exemple de cod sau șabloane de proiect.
* Standardizare: interfața IDE standardizează procesul de dezvoltare, care ajută programatorii să lucreze împreună mai ușor și ajută noii angajați să termine cu o viteză mai mare task-urile.

Există o varietate diversificată de IDE-uri, care se ocupă de moduri diferite în care lucrează dezvoltatorii și codurile sursă diferite pe care le produc. Există IDE-uri concepute pentru un singur limbaj de programare și IDE-uri care sunt proiectate să meargă pentru o listă mai mare de limbaje.

Printre cele mai cunoscute IDE-uri se află:

* Eclipse – suportă C, C++, Python, Perl, PHP, Java, Ruby
* NetBeans – suportă Java, Javascript, PHP, Python, Ruby, C, C++
* Komodo IDE – suportă Perl, Python, Tcl, PHP, Ruby, Javascript
* Aptana – suportă HTML, CSS, Javascript, AJAX
* Geany – suportă C, Java, PHP, HTML, Python, Perl, Pascal
* Visual Studio – suportă Visual C++, VB.NET, C#, F#
* MonoDevelop – suportă C, C++, Visual Basic, C# si alte limbaje .NET
* Xcode – suportă Objective-C, Swift si API-urile Cococa si Cocoa Touch
* Espresso – suportă HTML, CSS, XML, Javascript, PHP
* Coda – suportă PHP, Javascript, CSS, HTML, AppleScript si API-ul Cocoa
* CodeLite – suportă C, C++
* C-Free – suportă C, C++
* Jikes – suportă Java
* Idle – suportă Python
* RubyMine – suportă Ruby/Rails

## Motivație

* Even if you have a fast computer, chances are you might still need more computing power. Cloud computing offers you that, computing power at your fingertips and, better yet, you decide how much of it to use and when.
* Compiling remotely ensures that your workstation is free to do other computing intensive tasks, like encoding/decoding, testing applications, rendering or number crunching of any kind. Or even if you just need it smooth enough to use your everyday applications.
  + Additionally, sometimes you need to ensure that your program runs in different versions of the same platform, or different platforms and operating systems. Sure, virtualization can be done locally with a slew of software applications but this way, you don’t have to worry about setting up anything but the servers themselves. And lest not forget, they can be used for other tasks as well.
* There are arguably many cases where a remote compilation pays off, so let’s see how one would usually go about this.

# Abordări anterioare

# Descrierea soluției

Aplicația OnlineCompiler oferă programatorilor posibilitatea de a executa și testa cod scris în diverse limbaje fără a fi nevoie să-și instaleze compilatoarele specifice în calculatorul propriu. Aceștia vor putea astfel scăpa de problemele lipsei de spațiu, scăderii performanței și creșterea folosirii CPU.

## Arhitectura aplicației

Pentru crearea aplicației web, au fost folosite tehnologiile ce formează Mean stack. Acesta este o abordare modernă a dezvoltării web-ului în care un singur limbaj, Javascript, rulează în fiecare nivel al aplicației, de la client la server la persistență. Cele patru tehnologii care fac parte din acest grup și a căror inițiale formează acronimul Mean sunt: MongoDB, ExpressJS, AngularJS și Node.js.

### Front-end

Front-endul reprezintă partea vizuală a aplicației cu care utilizatorii interacționează. Principalele tehnologii folosite sunt HTML pentru definirea structurii întregului site, CSS pentru stilizarea și aranjarea în pagină a elementelor create în scheletul HTML, și Javascript pentru a defini interacțiunile care să atragă utilizatorii.

Interfața constă într-o pagină cu un aspect asemănător unui IDE, ce conține un editor de text în partea stângă, un file explorer și o casuță pentru rezultat în partea dreaptă. Design-ul dispune de 2 teme: una închisă și una deschisă așa cum oferă majoritatea IDE-urile. În partea de sus a editorului sunt mai multe butoane, inclusiv butonul care pornește execuția fișierului, butoane pentru mărirea sau micșorarea textului și de activare sau dezactivare a funcției de auto-complete.

Editorul de text este unul special care are un mod diferit pentru fiecare limbaj, astfel încât cuvintele cheie sunt colorate diferit. Din interfață se poate schimba modul editorului prin schimbarea limbajului din bara de sus a paginii, în acest fel modificându-se compilatorul sau interpretorul și comanda care se va folosi pentru a executa programul.

În partea dreaptă a paginii se află o ierarhie de fișiere. Inițial, există un singur director cu câte un fișier Hello World pentru fiecare limbaj. Utilizatorul poate să adauge fișiere sau directoare noi, să uploadeze din sistemul local un fișier, să downloadeze unul din fișierele de pe interfață și să redenumească sau să șteargă fișiere sau directoare.

La prima accesare a site-ului, se setează automat un cookie în browser care va fi verificat de fiecare dată când pagina este vizitată din nou. Serverul salvează cookie-ul într-o bază de date ca o sesiune ce conține și fișierele pe care le are utilizatorul. Astfel, de fiecare dată când pagina este reaccesată, fișierele nu vor fi pierdute deoarece starea este pastrată chiar și când aplicația este închisă pentru o perioadă de timp limitată.

### Back-end

Back-endul este partea de server a aplicației, cea pe care utilizatorii nu o pot vedea, și cea care face comunicarea dintre browser și baza de date. Acesta este constituit din trei părți: serverul, aplicația găzduită pe el și baza de date care împreună, așteaptă cereri de la codul ce rulează pe browser, cereri făcute de utilizatori prin intermediul interfeței, și trimite răspunsul dorit.

Tehnologiile principale în crearea aplicației de back-end sunt Node.js și framework-ul ExpressJS. Baza de date utilizată este una de tip NoSQL, și anume MongoDB. Pentru că clientul care folosește aplicația ar putea trimite o bucată de cod menită să șteargă fișiere importante de pe server, am folosit Docker, creând un container în care codul clientului rulează, păstrând astfel siguranța serverului.

#### Docker

Docker este o platformă software open-source creată pentru a rula aplicații într-un mediu izolat de restul sistemului. Acesta funcționează ca o mașină virtuală care ocupă mult mai puțin spațiu. Docker este un instrument ce împachetează, asigură și rulează containere în mod independent de sistemul de operare pe care este instalat.

“Definite printr-un singur fișier de configurare, containerele Docker pot fi refăcute de la zero în foarte scurt timp și vor funcționa identic indiferent de serverul pe care sunt refăcute. Este avantajoasă dezvoltarea aplicațiilor, local, folosind Docker pentru ca mai apoi să poată fi mutate în producție, fără prea multe surprize.” (Neagu, 2014)

O imagine de container este un pachet ușor, executabil și de sine stătător al unei bucăți de software ce include tot ce are nevoie pentru a o executa: cod, timp de rulare, instrumente și librării de sistem, setări. Fiind valabil și pentru Linux și pentru Windows, bucata de software conținută în container va rula în același mod de fiecare dată, indiferent de mediu. Containerele izolează software-ul de mediu, astfel încât diferențele dintre cadrul de dezvoltare și cadrul de producție nu vor mai conta și nu vor mai exista conflicte când mai multe echipe lucrează la același cod, deși au configurații diferite ale sistemelor.

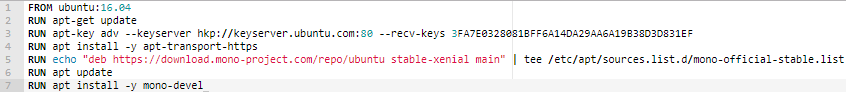


Fig. 3 – Dockerfile folosit pentru a crea o imagine de Docker pentru a executa C#

Pe serverul unde este urcată aplicația, sunt create imagini de Docker pentru fiecare limbaj de programare. Imaginile sunt bazate pe imaginea publică de Ubuntu la care se adaugă compilatorul și celelalte instrumente necesare. Pentru a construi o imagine de Docker, am creat un fișier numit Dockerfile, al cărui conținut este asemănător cu cel din Fig. 3. Comanda folosită pentru a crea imaginea numită c-sharp este (se execută din interiorul directorului care conține fișierul Dockerfile):

*docker build –t c-sharp .*

Pentru fiecare din limbajele implementate în interfață, există o imagine diferită în care au fost instalate pachetele corespunzătoare limbajului, astfel:

* Pentru C – pachetul gcc
* Pentru C++ - pachetul g++
* Pentru C# - pachetul mono-devel
* Pentru Java – pachetul openjdk-8-jdk
* Pentru Python 2 si 3 – pachetul python

Atunci când utilizatorul vrea să execute o bucată de cod, front-endul o trimite spre back-end de unde serverul va crea un container de Docker bazat pe o nouă imagine derivată de la cea originală a limbajului ales dar care are în plus fișierul trimis în sistem. De exemplu, pentru o cerere de la client de a executa un HelloWorld in C#, scriptul în Node.js va crea automat un fișier Dockerfile ce conține textul din Fig. 4.



Fig. 4 – Dockerfile pentru execuția unui program Hello.csc scris in C#

Aplicația fiind destinată celor care vor să învețe un limbaj nou sau să testeze bucăți de cod de mici dimensiuni, nu poate executa un script la infinit. De asemenea, există cazuri în care un utilizator poate implementa o aplicație care să asculte la un anumit port, caz în care execuția nu se va termina decât în cazul în care crapă sau alte situații definite în cod. În astfel de cazuri, am limitat timpul în care se așteaptă răspunsul, folosind argumentul *timeout* din cadrul comenzii de începere a container-ului Docker:

*timeout –signal KILL 60 docker run docker\_image*

În cazul în care scriptul nu a terminat de rulat, se va trimite semnalul KILL și el se va opri forțat. Clientul va primi în răspuns faptul că s-a ajuns la timpul maxim de execuție.

La sfârșitul execuției, după recuperarea output-ului, container-ul în care aplicația a fost rulată va fi șters. Dacă mai mulți utilizatori accesează pagina în același timp, stările lor vor fi independente una de alta iar fișierele vor fi executate în containere diferite care nu au nicio legătura între ele.

## Google Cloud

Google Compute Engine este o componentă IaaS din platforma Google Cloud ce permite utilizatorilor să își creeze mașini virtuale în care să-și execute aplicațiile. Mașinile virtuale se deschid repede, au disc de stocare persistent și livrează performanță înaltă consistentă. Google Cloud permite utilizatorului să iși configureze VM-ul, putând opta pentru imaginile implicite dar și având opțiunea să iși creeze mașina virtuală după o imagine particularizată de Docker sau alte tehnologii de containizare.

Odată creată mașina virtuală, interfața Google Cloud permite accesul în mașină prin SSH direct din browser. Aici mediul poate fi pregătit pentru a putea rula aplicația, instalând toate programele necesare cu ajutorul comenzilor specifice sistemului de operare.

În VM-ul creat, am instalat Node.js și Nginx care permite aplicației să fie accesată la portul 80 și am pornit aplicația folosind pachetul pm2, un manager de procese pentru deployment ce oferă management de loguri, politici de repornire automată, monitorizare pentru aplicație, etc. În cadrul Google Cloud, este posibilă de asemenea configurarea unui snapshot de VM și setări de aplicație astfel încât aplicația să se scaleze automat. Astfel, în cazul în care interfața primește într-o perioadă scurtă de timp un număr mult mai mare de cereri față de cum era obișnuită, Google Cloud mărește numărul de instanțe de mașini pe care să ruleze aplicația ca să poată face față tuturor cererilor.

# Concluzii

# Bibliografie

*Why the Hell Would You Use Node.js*. (2017, 2 24). Preluat pe 6 14, 2018, de pe medium: https://medium.com/the-node-js-collection/why-the-hell-would-you-use-node-js-4b053b94ab8e

*Docker Containers*. (fără an). Preluat pe 6 11, 2018, de pe aquasec: https://www.aquasec.com/wiki/display/containers/Docker+Containers

Ferguson, A. (2000). *A History of Computer Programming Languages*. Preluat pe 6 19, 2018, de pe cs.brown.edu: https://cs.brown.edu/~adf/programming\_languages.html

IBM Corporation. (fără an). *Compiled versus interpreted languages*. Preluat pe 6 20, 2018, de pe ibm: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zappldev/zappldev\_85.htm

Jackson, B. (2018, 5 28). *Top 7 Advantages of Choosing Google Cloud Hosting*. Preluat pe 6 15, 2018, de pe kinsta: https://kinsta.com/blog/google-cloud-hosting/

Neagu, C. L. (2014, 8 21). *Îndrumar Docker*. Preluat pe 6 10, 2018, de pe clneagu: http://clneagu.ro/docker/

*Node.js - Introduction*. (fără an). Preluat pe 6 14, 2018, de pe tutorialspoint: https://www.tutorialspoint.com/nodejs/nodejs\_introduction.htm

Sahoo, P. (2015, 4 26). *Basic concepts in AngularJS*. Preluat pe 6 9, 2018, de pe c-sharpcorne: https://www.c-sharpcorner.com/UploadFile/dev4634/basic-concepts-in-angular-js/

thussong. (2015, 7 30). *Programming Languages Through the Years*. Preluat pe 6 19, 2018, de pe thesoftwareguild: https://www.thesoftwareguild.com/blog/history-of-programming-languages/

Ularu, E.-G., & Puican, F. (2012, 12). *NOUA GENERAŢIE DE BAZE DE DATE NoSQL*. Preluat pe 6 10, 2018, de pe rria: https://rria.ici.ro/wp-content/uploads/2012/12/06-art.-Ularu-si-Puican-revizuit.pdf

1. Electronic Numerical Integrator and Computer [↑](#footnote-ref-1)
2. Permite scrierea unor programe care să fie independente de tipul calculatorului pe care rulează [↑](#footnote-ref-2)
3. Poziția caracterelor în fișier nu este relevantă [↑](#footnote-ref-3)
4. https://www.codecademy.com/ [↑](#footnote-ref-4)
5. https://leetcode.com/ [↑](#footnote-ref-5)
6. https://www.hackerrank.com/ [↑](#footnote-ref-6)
7. https://projecteuler.net/ [↑](#footnote-ref-7)
8. https://stackoverflow.com/ [↑](#footnote-ref-8)
9. https://codereview.stackexchange.com/ [↑](#footnote-ref-9)