**TITLUL LUCRĂRII DE LICENŢĂ**

LUCRARE DE LICENŢĂ

|  |  |  |
| --- | --- | --- |
|  | Absolvent: | **Popirtan Vlad-Mihai** |
|  |  |  |
|  | Coordonator științific: | **Prof. Dr. Ing. Mădălin Neagu** |

**2022**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
| DECAN, |  | | DIRECTOR DEPARTAMENT, | |
| **Prof. dr. ing. Liviu MICLEA** |  | **Prof. dr. ing. Rodica POTOLEA** | |

Absolvent: **Popirtan Vlad-Mihai**

**TITLUL LUCRĂRII DE LICENŢĂ**

1. **Enunțul temei:** *Scurtă descriere a temei lucrării de licență și datele inițiale*
2. **Conținutul lucrării:** *(enumerarea pârților componente) Exemplu: Pagina de prezentare, aprecierile coordonatorului de lucrare, titlul capitolului 1, titlul capitolului 2,… titlul capitolului n, bibliografie, anexe.*
3. **Locul documentării**: *Exemplu*: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți**:
5. **Data emiterii temei:** 1 noiembrie 2021
6. **Data predării:** 8 iulie 2022

|  |  |
| --- | --- |
| Absolvent: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |
| Coordonator științific: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Declarație pe propria răspundere privind**

**autenticitatea lucrării de licență**

Subsemnatul(a)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**, legitimat(ă) cu \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ seria \_\_\_\_\_\_\_ nr. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
CNP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, autorul lucrării \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ din cadrul Universității Tehnice din Cluj-Napoca, sesiunea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ a anului universitar \_\_\_\_\_\_\_\_\_\_, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

In cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

|  |  |  |
| --- | --- | --- |
| Data  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Nume, Prenume  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  | Semnătura |

**Instrucțiuni generale**

**De citit înainte** (paginile de aici până la cuprins se vor elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declarație) se vor lista pe foi separate (nu fată-verso), fiind incluse în lucrarea listată.

Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului.

Pe declarație se trece data când se predă lucrarea la secretarii de comisie.

1. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător (consultați pagina de unde ați descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).
2. **Cuprinsul** începe pe pagina nouă, impară (dacă se face listare fată-verso).

Toate capitolele încep pe o pagină nouă. Numerotarea paginilor începe de la 1 la primul capitol (**Introducere)**. Pentru actualizarea cuprinsului, folosiți meniul *References: Table of Contents->Update table*.

1. Vizualizați (și în timpul editării, dacă e cazul) acest document după ce activați vizualizarea simbolurilor ascunse de formatare (click pe pictograma **** din *Home/Paragraph*).
2. Fiecare capitol începe pe pagină nouă. Acest lucru este realizat folosind simbolul de formatare *Section Break* cu opțiunea *Next Page*, care a fost folosit la sfârșitul fiecărui capitol. Dacă ștergeți din greșeală simbolul, îl puteți pune înapoi folosind submeniul *Breaks* al meniului *Layout*.
3. Folosiți stilurile predefinite (Headings, Figura, Tabel, Normal, etc.)
4. Mărimea hârtiei este A4, iar marginile paginilor sunt 25.4 mm toate.
5. Respectați restul instrucțiunilor din fiecare capitol.
6. Înainte de salvarea ca fisier pdf, alegeți din meniul *Options* opțiunile din figura următoare



Cuprins

[Capitolul 1. Introducere – Contextul proiectului 1](#_Toc66962577)

[1.1. Contextul proiectului 1](#_Toc66962578)

[1.1.1. Exemplu de titlu subsecțiune 1](#_Toc66962579)

[Capitolul 2. Obiectivele Proiectului 3](#_Toc66962580)

[Capitolul 3. Studiu Bibliografic 5](#_Toc66962581)

[Capitolul 4. Analiză și Fundamentare Teoretică 7](#_Toc66962582)

[Capitolul 5. Proiectare de Detaliu si Implementare 9](#_Toc66962583)

[Capitolul 6. Testare și Validare 11](#_Toc66962584)

[Capitolul 7. Manual de Instalare si Utilizare 13](#_Toc66962585)

[Capitolul 8. Concluzii 15](#_Toc66962586)

[Bibliografie 17](#_Toc66962587)

[Anexa 1 19](#_Toc66962588)

# Introducere – Contextul proiectului

Tehnologia este un aspect ce evoluează din ce in ce mai rapid in zilele noastre, lucru ce influențează in mod direct viața de zi a oamenilor. Deși exista si aspecte negative ale acestei evoluții, trebuie sa amintim ca tehnologia aduce in principal beneficii asupra noastră. Multe lucruri ce in trecut păreau imposibile, precum comunicarea zilnica cu prietenii sau familia de la distanta, acum se pot realiza cu ajutorul câtorva apăsări de butoane prin intermediul apelurilor telefonice, sau prin folosirea rețelelor de socializare.

Un alt rezultat al acestei evoluții este automatizarea industriala. Prin înlocuirea oamenilor cu roboti industriali, se elimina, sau cel puțin diminuează, probleme precum riscul erorilor umane de calcul, riscul accidentarilor in timpul operațiilor, inconsistenta adusa de stările diferite in care se poate afla o persoana umana, sau nevoia de buget crescut alocat salariilor angajaților. Cantitatea produsa de industrii a devenit si ea mai mare odată cu automatizarea proceselor, robotii industriali neavând nevoie de pauze, aceștia fiind funcționali pe tot parcursul zilei.

## Contextul proiectului

Deși am precizat ca folosirea roboților industriali si automatizarea sarcinilor reduce nevoie de personal uman, întrebarea care a apărut este „Cine creează această automatizare?”. Astfel a apărut domeniul IT. Domeniul IT a devenit unul foarte căutat de studenți din mai multe motive:

* copii sunt atrași de calculator încă de la o vârstă frageda, datorita jocurilor video;
* este un domeniu dificil, perfect pentru persoanele dornice de noi provocări, fie acestea create de necunoscutul tehnologiei, sau de dorința de performanta;
* Este recunoscut ca unul din cele mai bune plătite domenii in care poți activa;

Așa ca orice persoana poate fi atrasa de ideea de a deveni programator. Însă principala calitate pe care trebuie sa o dețină un programator este gândirea algoritmica. Deși nu este un lucru dificil sa înveți un limbaj de programare, sau sa urmezi niște pași care sa te duca la îndeplinirea sarcinii, acest lucru nu te va duce prea departe in cariera ta, atat pe plan financiar, cat si in propria satisfacție. Vor veni momente in care pentru a te face remarcat trebuie sa fii in stare sa gândești un sistem de la zero, sau sa aduci o soluție ce va beneficia produsul, ridicându-l deasupra competiției.

Intr-un studiu realizat folosind Stack Overflow Jobs [1], reiese ca pe lângă cunoștințele in domeniul programării (ex: Limbajul de programare, librarii, unelte de development), companiile sunt interesate de așa zisele soft skills, caracteristicile unui individ legate de comportamentul acestuia (ex: Comunicare, lucrul in echipa, abilitățile de lider). In urma analizării descrierii a douăzeci de mii de postări de recrutare, pe lângă abilitățile de comunicare si abilitați legate de colaborare, cuvintele „analytical”, „problem solving” sau „deliver” sunt prezente in una din zece postări, cuvinte ce denota abilitatea de rezolvare a problemelor menționată mai sus.

O alta calitate pe care trebuie sa o aibă un programator este abilitatea de a lucra rapid. Cum nu fiecare proces este menit succesului din prima încercare, un programator bun trebuie sa fie in stare sa încerce mai multe soluții pentru problema in cauza, lucru care ar lua foarte mult timp in cazul unui programator lent.

Datorita celor menționate mai sus, putem trage concluzia ca orice persoana poate deveni un programator bun, atât timp cat este dispus sa sacrifice timpul necesar pregătirii acestor cerințe. Însă calitățile prezentate se pot dobândi doar prin exercițiu. Abilitatea de a rezolva probleme se poate antrena prin abordarea a diferitor probleme, in timp ce abilitatea de a scrie cod rapid si eficient este doar ajutata de practicarea regulata a programării.

Pentru a expune copiii la domeniul IT, scoli si licee au introdus materia de de calcul in programa școlară, inca din primii ani ai elevului. Anglia a introdus materii de calcul chiar din grădiniță, in Australia copii sunt expuși la tehnologii digitale din primul an școlar, in timp ce in Finlanda, materii de calcul sunt introduse încă din primul an de școală [2].

## Problema

O mare parte din întrebările de interviu pentru companiile software, este reprezentata de probleme de algoritmica, pentru a observa gândirea candidatului. Deși școlile fac eforturi sa introducă materii de informatica in programa școlară, timpul limitat care este alocat acestor materii, nu poate pregăti un elev pentru așteptările companiilor atunci când recrutează.

După cum am menționat mai sus, prin pachetul salarial atractiv pe care îl oferă acest domeniu, programarea poate stârni interesul unor oameni ce nu au terminat o facultate de specialitate. Sunt oameni ce nu doresc sa investească anii necesari pentru a termina o facultate de specialitate, iar lipsa de resurse educaționale ii poate descuraja din a afla o posibila pasiune din care sa câștige si un trai. La fel ca orice domeniu, exista manuale si cursuri online, însă acestea nu conțin o diversitate ridicata, se axează in principiu pe conceptele de baza ale programării si nu pun accentul pe optimizarea soluțiilor.

## Abordarea problemei

Titlul proiectului reprezintă o platforma online, gratuita si valabila oricui dorește sa își testeze abilitățile de programare. Fiecare soluție va fi rulata pe un set de teste, specifice fiecărei probleme, ce asigura corectitudinea completa a soluției, investigând situații specifice, la care utilizatorul va fi nevoit sa gândească suplimentar pentru a le găsi si trata.

Cum fiecare persoana este familiara cu internetul si aplicațiile web, titlul proiectului dorește să facă utilizarea sa plăcută pentru orice utilizator, acesta nefiind nevoit sa instaleze diferite compilatoare sau aplicații suplimentare pentru a isi testa soluția la problema. Utilizatorul va beneficia de un editor de text, integrat in pagina aplicației, care sa se ocupe de funcționalitatea de „syntax highlighting”.

Fiecare soluție va fi verificata din punct de vedere al corectitudinii, atât in cadrul rezolvării problemei propuse, dar si din punct de vedere sintactic. Aceasta va fi verificata si din punct de vedere al timpului de rulare si al memoriei utilizate, fiecare problema având aceste constrângeri predefinite.

Orice utilizator poate rezolva problema aleasa, in limbajul sa de programare preferat, atata timp cat acesta se afla in optiunile valabile: Java, Python, C/C++.

# Obiectivele Proiectului

Obiectivul acestui proiect este de a crea o platforma, in care se poate construi o comunitate de oameni pasionați de informatica, care doresc sa își testeze abilitățile zilnic într-un mod gratuit.

## Cerințe funcționale

Aplicația dorește sa reprezinte o platforma web, care sa poată fie plăcută utilizatorilor atunci când o accesează, de aceea aceasta trebuie sa prezinte un pachet complet de funcționalități:

* **Probleme de liceu** - Aplicația trebuie sa conțină o colecție de probleme de liceu, pentru a putea permite elevilor ce doresc sa urmeze o facultate de domeniu sa se pregătească pentru examenul de bacalaureat, sau pentru olimpiade de informatica. Totodată, putem considera acest nivel de probleme ca fiind cel introductiv. Aceste probleme nu sunt menite doar elevilor de profil matematica-informatica, aceste probleme pot reprezenta punctul de start pentru cineva nou in acest domeniu
* **Probleme de facultate** – Aplicația trebuie sa conțină o colecție de probleme de facultate, unde studenții se pot pregăti cu noile cunoștințe dobândite in cadrul facultății (ex: Liste, grafuri, pointeri si referințe). Aceste probleme reprezintă un nivel mai avansat fata de cel de liceu, nu neapărat al dificultății lor, ci al bagajului de cunoștințe pe care trebuie sa îl aibă utilizatorul înainte de a încerca să găsească o soluție.
* **Probleme de interviu** – Aplicația trebuie sa conțină probleme ce sa pregătească utilizatorul pentru interviuri in cadrul companiilor. Majoritatea interviurilor conțin măcar o problema de algoritmica. Astfel aplicația trebuie sa conțină o colecție variata de probleme întâlnite de alți utilizatori pentru a ajuta alți membrii ai comunității sa se pregătească.
* **Posibilitatea de a adăuga probleme noi –** Pentru a ajuta comunitatea, fiecare utilizator trebuie sa aibă posibilitatea de a adăuga probleme întâlnite in diferite scenarii. Cu ajutorul acestei funcționalități, colecția de probleme va deveni una completa, un utilizator care dorește sa se pregătească fiind nelipsit de opțiuni si diversitate
* **Un editor de text integrat in browser –** Un editor de text direct in browser este modul aplicației de a facilita procesul de trimitere al unei soluții. Utilizatorul nu este nevoit sa își instaleze propriul editor de text, TITLU punând deja la dispoziție unul ce conține si syntax highlighting pentru un aspect vizual plăcut;
* **Multiple limbaje de programare –** Deoarece învățarea unui limbaj de programare, nu reprezintă scopul acestei aplicații, si cum orice problema poate fi rezolvata in oricare dintre cele mai populare limbaje, aplicația nu ar fi una completa daca ar forța utilizatorul sa învețe un anumit limbaj pentru a trimite soluții. Astfel utilizatorului trebuie sa poată trimite o rezolvare in orice limbaj de programare ii este confortabil.
* **Limite de memorie si timp –** Fiecare soluție va fi supusa mai multor teste ce vor fi verificate din punct de vedere al timpului si al memoriei utilizate. Aceasta cerința este necesara pentru a încuraja utilizatorul sa găsească soluții optime la probleme. Daca soluția propusa nu se încadrează in limitele de timp pentru unul din teste, aceasta nu va primi punctaj pe problema respectiva.

## Cerințe non-funcționale

Se dorește ca aplicația sa ii ofere utilizatorului o experiență plăcută pe tot parcursul timpului petrecut, așa ca avem nevoie de un set de cerințe non-funcționale pe care sa le respectam:

* **Scalabilitate** – Fiind o aplicație online, este de așteptat sa întâmpinăm un număr mare de utilizatori concurenți sau o baza de date încărcată. Acest lucru nu trebuie sa afecteze aplicația in așa fel încât aceasta nu devina neutilizabila. Ea trebuie sa accepte un număr mare de accesări si sa ii fie la fel de ușor sa găsească datele necesare atât pentru o baza de date încărcată, cat si pentru una cu puține intrări;
* **Securitate** – Aplicația trebuie sa fie securizata din punct de vedere al datelor utilizatorului. Funcționalitățile aplicației nu vor funcționa in cazul unui utilizator neautentificat.
* **Utilizabilitate** – Aplicația trebuie sa fie ușor de învățat pentru orice utilizator. Cum <> dorește sa fie de folos atât unor persoane ce au de a face cu domeniul IT, fie printr-o facultate de profesie sau prin pura pasiune, aceasta este menita sa învețe, așa ca trebuie sa poată fi folosita de oameni cu mai puține cunoștințe.
* **Capacitate** – Datorita nivelului mare de date ce va fi stocat de către aplicație, aceasta trebuie sa aibă implementat o logica de stocare a datelor in diferite locații
* **Integritatea datelor** – Toate datele trebuie sa respecte reguli impuse de programator, pentru a asigura ca toate datele sunt interpretabile de către aplicație, sau ca asigura siguranța utilizatorului
* **Testabilitate** – Aplicația trebuie sa fie împărțite in module, astfel încât fiecare modul sa poată fi testat separat. Astfel putem descoperi mai rapid de unde apar anumite probleme, si repararea lor este la rândul ei mai rapida
* **Performanta** – După realizarea unei operații, utilizatorul trebuie sa poată naviga in continuare aplicația, fără a aștepta prea mult după răspunsul acesteia
* **Extensibilitate** – Fiind o aplicație web menita utilizatorilor, aceasta trebuie sa fie capabila de a fi extinsa in viitor pentru a satisface nevoile oamenilor. Ea trebuie sa permită introducerea de noi funcționalități, păstrând funcționarea celor vechi
* **Valabilitate** – Fiind o aplicație web, aceasta trebuie sa fie funcțională majoritatea timpului. Putem planifica anumite date si ore in care sa aibă loc o resetare a sistemului(ex: O data pe săptămâna), însă aceste resetări ar fi făcute într-o perioada in care aplicația înregistrează cele mai puține accesări
* **Fiabilitate** – Chiar daca o operație nu este îndeplinită intr-un mod corect, aplicația nu trebuie sa ajungă într-o stare fatala. Aceasta trebuie sa poată transmite printr-un mesaj de eroare utilizatorului, ca ceva nu a mers corect si sa verifice datele trimise.

## Adăugarea unei probleme – Caz de utilizare

### Descriere

In acest caz de utilizare este prezentată operația de propunerea unei noi probleme de către un utilizator. Acesta va adăuga toate datele necesare problemei: titlu, enunț, un exemplu de date de intrare si de ieșire, timpul si memoria limita, si de asemenea setul de teste.

### Actorul principal

Actorul principal in cadrul acestei operații este orice utilizator al aplicației. Pentru încurajarea ajutării comunității, fiecare utilizator trebuie sa fie in stare sa propună o problema noua întâlnită.

### Scenariul de baza

**Start caz de utilizare**

Acest caz de utilizare începe atunci când un utilizator dorește sa adauge o noua problema in lista

1. Utilizatorul selectează fereastra de problema, apăsând butonul aferent acestei operații.
2. Utilizatorul accesează fereastra de introdus date ale problemei, apăsând butonul de adăugare a unei probleme, sugerata de o pictograma sugestiva.
3. Utilizatorul introduce datele problemei completând câmpurile sugerate de o eticheta
4. Utilizatorul încarcă setul de teste din propriul sau calculator, apăsând butonul de „Încărcare” si selectând fișierele de teste.

**Final caz de utilizare**

Utilizatorul decide sa finalizeze operația, apăsând butonul de „Save” din interfața utilizator, salvând problema.

### Scenarii alternative

* Utilizatorul se oprește din operația de salvare a problemei, accesând din nou lista de probleme prin apăsarea butonului aferent.
* Utilizatorul poate reîncărca setul de teste, cel precedent fiind suprascris, reapăsând butonul de „Încărcare”

### Precondiții

Pentru ca operația sa fie realizabila, utilizatorul trebuie sa se asigure de un număr de precondiții înainte de a începe:

* Utilizatorul trebuie sa fie logat in aplicatie cu un set de credentiale valide.
* Utilizatorul trebuie sa se asigure ca are un număr egal de fișiere de intrare si de referința.
* Utilizatorul trebuie sa se asigure ca fișierele de teste sunt denumite după următorul format:

1. Pentru fișiere de intrare: *TitluProblema\_NumeTest.in*
2. Pentru fișiere de referință: *TitluProblema\_NumeTest.ref*

* Utilizatorul trebuie sa se asigure ca nu exista deja o problema cu același titlu, ca cel al căreia el dorește sa o încarce

Diagram

Description automatically generated

Figură 2.1

## Propunerea unei solutii – Caz de utilizare

### Descriere

In acest caz de utilizare este prezentata operatiunea de propunere a unei solutii de catre utilizator. Acesta isi va alege problema la care doreste sa propuna o solutie, limbajul de programare in care doreste sa rezole si intr-un final va putea trimite spre verificat solutia

### Actorul principal

Actorul principal este orice utilizator al aplicatiei. Scopul principal al platformei este antrenarea cunostintelor de algoritmica, asa ca acesta este un caz de utilizare specific pentru toti utilizatorii.

### Scenariul de baza

**Start caz de utilizare**

Acest caz de utilizare incepe atunci cand un utilizator isi alege o problema pe care sa o rezolve, si dodreste sa propuna o solutie pentru aceasta.

1. Utilizatorul selecteaza fereastra de problema, apasand butonul aferent acestei operatii.
2. Utilizatorul isi alege o problema din lista, si acceseaza pagina acesteia folosind butonul „Try”, prezent in dreaptul fiecarei probleme
3. Utilizaotrul isi alege limbajul de programare in care doreste sa rezolve problema, folosind un dropdown.
4. Utilizatorul introduce solutia pentru problema propusa in editorul de text prezent pe pagina problemei

**Final caz de utilizare**

Utilizatorul decide sa finalizeze operatiunea prin apasarea butonului de „Submit” din interfata de utilizator, trimitand astfel solutia spre verificare pe setul de teste.

### Scenarii alternative

* Utilizatorul poate renunța la aceasta operație, apăsând butonul „Înapoi”, reprezentat printr-o pictograma sugestiva.
* Utilizatorul poate schimba oricând limbajul de programare in care scrie, resetând codul la un schelet specific limbajului ales.

### Precondiții

Pentru ca operația sa fie realizabila, utilizatorul trebuie sa se asigure de un număr de precondiții înainte de a începe:

* Utilizatorul trebuie sa fie logat in aplicație cu un set de credențiale valide.

Diagram

Description automatically generated

Figură 2.2

## Vizualizarea rezultatului unei soluții – Caz de utilizare

### Descriere

In acest caz de utilizare este prezentata operatiunea de vizualizare a statusului unei solutii. Utilizatorul isi poate vedea toate solutiile propuse, vazand statusul lor *SUCESSFULL/FAILED,* si sa vizualizeze detalii legate de aceasta solutie cum ar fi timpul de executie, limbajul de programare in care a fost scrisa, cat si solutia propruzisa intr-o fereastra de cod.

### Actorul principal

Actorul principal este orice utilizator care a trimis o solutie catre o problema din cele puse la dispozitie de catre aplicatie. Scopul aceste platforme este unul de a incuraja gasirea solutiilor cat mai optime din punct de vedere al timpului si al memoriei, dar si gasirea unei solutii complete, care sa rezolve toate posibilitatile problemei in cauza. Utilizatorul poate vizualiza atat statusul fiecarui test, cat si timpul de executie necesar pentru rezolvarea testului respectiv.

### Scenariul de baza

**Start caz de utilizare**

Acest caz de utilizare incepe atunci cand utlizatorul doreste sa verifice statusul unei solutii propuse.

1. Utilizatorul selecteaza fereastra de problema, apasand butonul aferent acestei operatii.
2. Utilizatorul alege o submisie careia sa ii vizualizeze detaliile
3. Utilizatorul selecteaza testele, pentru a le inspecta statusul si detalii legate de rulare

### Scenarii alternative

Nan

### Preconditii

Pentru ca operația sa fie realizabila, utilizatorul trebuie sa se asigure de un număr de precondiții înainte de a începe:

* Utilizatorul trebuie sa fie logat in aplicație cu un set de credențiale valide.

Diagram

Description automatically generated

Figură 2.3

# Studiu Bibliografic

Nevoie de o platformă de online de învățare a fost cel mai mult vizibila după evenimentele întâmplate in 2020, si anume impactul virusului COVID-19 asupra populației. Educația a fost una dintre cele mai afectate ramuri, in urma nevoii de distanțare socială, școlile au fost nevoite sa își suspende activitatea fata in fata. In urma unui studiu realizat de Organizația Salvați Copiii Romania, reiese ca plictiseala este principala stare resimțita de copii (47.5%), urmata de oboseala[3]. Sentimentele precizate sunt cauzate de lipsa de interacțiune intre profesori si elevi, multe cadre didactice continuând sa își susțină orele in aceeași maniera ca in afara contextului pandemic, iar elevii preferând sa ia parte in alte activități in timpul orelor de curs, același studiu arătând ca 57.7% din copii prefera sa se joace pe telefon, tableta sau calculator[3].

Deși contextul pandemic a avut un evident impact asupra tuturor ariilor ale educației, printre aceasta se include si programarea, o materie care da impresia ca s-ar putea învăța individual fără probleme. Acest lucru a fost dovedit contrar, in urma unui studiu, 73%, dintre care 58% fiind studenți in domeniu, din cei intervievați au răspuns ca au întâlnit probleme in învățarea programării, datorita lipsei unui profesor sau mentor ce sa ii ajute in instalarea mediului de învățare. Aceștia considera ca trecerea la învățarea in mediul online reprezintă un stres ce fără îndoială va avea un impact asupra rezultatului final. Figura 3.1 reprezintă un grafic al problemelor întâlnite de către studenți in timpul scolii online[4].

Chart

Description automatically generated

Figură 3.1

In [5], a fost descoperit ca folosirea simulatoarelor online pentru a își antrena abilitățile de programare, a simplificat si procesul de predare al profesorilor. Pentru aceștia este mult mai ușor sa îi trimită unui elev o problema prin intermediul aplicației, decât sa citească codul, sa identifice manual erorile si sa găsească lacunele ce îl împiedică pe elev in cariera sa.

Intr-un alt studiu [6] este evidențiat cum predarea programării către elevi poate prezenta foarte multe piedici. Programarea este un subiect costisitor din punct de vedere al timpului pentru a fi învățat, este un domeniu vast ce poate părea copleșitor, prin urmare este greu sa ții elevei motivați in tot timpul necesar sa învețe si sa se pregătească prin procesul de eroare si eșec sa învețe programare. Din acest motiv au fost create platforme online de coding care sa transforme procesul de programare într-un joc. Prin interfețe interactive si plăcute vizual, reușesc sa atragă atenția utilizatorului. Aceste platforme au rolul de a învăța utilizatorul bazele programării pentru cei noi in acest domeniu, dar prezinta si provocări pentru cei deja experimentați. Câteva aspecte ce fac folosirea acestor unelte atractive sunt[4]:

* Sunt valabile oricui, nefiind necesara instalarea unor unelte suplimentare, resurse multe ale calculatorului sau un sistem prea modern. O astfel de platforma poate fi accesata de către un utilizator pe un calculator sau telefon proaspăt cumpărat.
* Utilizatorii nu au nevoie de nimic mai mult decât o conexiune la internet, cea pusa la dispoziție de serviciul de telefonie mobila fiind suficienta. O alta precondiție pe care trebuie sa o îndeplinească utilizatorul este sa aibă in posesie un dispozitiv pe care sa poată rula o pagina web. Deși un laptop sau calculator facilitează cu mult operația, aceste platforme web pot fi accesate si de pe dispozitive mobile cum ar fi telefoane sau tablete.
* Prezenta exemplelor ajuta mult la înțelegerea problemei de către utilizatori. Aceștia nu trebuie sa vina cu un bagaj foarte mare de cunoștințe doar pentru a înțelege problema, platforma fiind destul de vizuala, astfel utilizatorul poate înțelege cerința problemei in timp ce scrie cod pentru a rezolva exemplul dat.
* Platformele de coding returnează un feedback instant al soluției, astfel salvează din timpul profesorului de a verifica soluții, a corecta erori si a încerca sa găsească problemele pe care le prezinta studentul. Asta ii dăruiește profesorului mai mult timp pentru a se axa pe materialele sale de predare online.

## Solutii existente

Aceste platforme de coding nu reprezintă un lucru nou pe piață, abordări existente fiind deja implementate:

### **HackerRank**

HackerRank este o platforma web, ce vine in ajutorul oamenilor cu niște cunoștințe deja existente in domeniul programării. Momentan HackerRank este liderul pe piață atunci când vine vorba de platforme de coding. Prin dificultatea problemelor prezente pe aceasta platforma, ea reprezintă prima alegere a companiilor atunci când își aleg noii angajați, asigurându-se ca noi recruții sunt stăpâni pe elementele de baza ale programării. In același timp, HackerRank este o platforma predominata de probleme de interviu, astfel aceasta este menita utilizatorilor ce au o oarecare baza pusa bine la punct de programare, si vor sa se pregătească pentru o viitoare cariera in acest domeniu. HackerRank suporta momentan 43 de limbaje de programare, un număr foarte atractiv pentru companii si de o vasta colecție de probleme de cod, care se întind pe mai multe domenii alte IT-ului (Ex: Programare Funcțională, algoritmica, SQL, AI etc.) Aceasta platforma reprezintă un loc bun de a învăța noi limbaje de programare, odată ce ai bazele acestui domeniu puse la punct. HackerRank este platforma cel mai mult bazata pe cariera, oferind si certificate in urma unui set de întrebări.[7]

A screenshot of a computer

Description automatically generated

Figură 3.2

### **LeetCode**

LeetCode este o alta platforma web menita sa ajute oamenii pasionați de programare sa isi testeze cunoștințele si sa își îmbunătățească abilitățile in acest domeniu. Aceasta platforma prezinta o vasta colecție de probleme de algoritmica, susținând doar 14 dintre cele mai populare limbaje de programare cum ar fi: Java, C, C++ si Python. Problemele sunt împărțite pe dificultate si se bazează in principal pe structuri de date si algoritmi. Spre deosebire de HackerRank, LeetCode conține si un pachet premium, unde se pot regăsi întrebări frecvente de interviu, structurate pe companii gigant in acest domeniu cum ar fi: Google, Facebook, Amazon, Microsoft etc. Leetcode prezinta si o funcționalitate de învățare, in care poti selecta diferite categorii, primele paginii fiind dedicate învățării, apoi utilizatorul va fi trimis catre un editor de text pentru a încerca sa pună in practica cele învățate.

Graphical user interface, text, application

Description automatically generated

Figură 3.3

### **Coderbyte**

CoderByte este încă o aplicație dedicata programatorilor ce doresc sa își pună la încercare abilitățile. Aceasta platforma ii permite utilizatorului sa rezolve probleme de algoritmica în 15 limbaje de programare, printre care: Python, C, C++, SQL etc. Coderbyte acorda utilizatorilor puncte pentru fiecare soluție trimisă, satisfăcând spiritul de competivitate din utilizatori. Ceea ce face Coderbyte unic fata de celalalte platforme menționate, este faptul ca aceasta permite utilizatorului sa vadă soluții trimise de alți utilizatori.

Text

Description automatically generated

Figură 3.4

### **Infoarena**

Infoarrena este o platforma web românească de rezolvat probleme de informatica, menita elevilor si studenților ce doresc sa învețe programare. Aceasta platforma este locul ideal pentru elevei ce doresc sa se pregătească de concursuri si olimpiade de informatica, însă nu prezinta probleme prezente in interviuri pentru companii din domeniu. Astfel Infoarena fata de cele prezentate mai sus, nu este o platforma dedicata celor ce si-au terminat studiile si doresc sa se pregătească pentru o viitoare cariera, ci mai mult pentru cei pasionați de acest domeniu. O problema pe care o prezinta acest site, este lipsa unui editor de text in browser. Pentru a putea testa o soluție, inițial trebuie instalat un editor de text, salvat soluția iar apoi încărcata.

Graphical user interface, text, application

Description automatically generated

Figură 3.5

## Solutia aleasă

In toate soluțiile existente menționate, toate erau menite către o singura audienta. Proiectul ales are scopul de a veni in ajutorul atât utilizatorilor experimentați prin probleme de interviu, atât cat si celor noi ce doresc sa își pună bazele programării.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | HackerRank | Leet  Code | Coder  byte | Info  arena |  |
| Suport pentru mai multe limbaje de programare | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| Editor in browser | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Close with solid fill | Checkmark with solid fill |
| Posibilitatea de a adauga probleme | Close with solid fill | Close with solid fill | Close with solid fill | Close with solid fill | Checkmark with solid fill |
| Probleme de interviu | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Close with solid fill | Checkmark with solid fill |
| Probleme de liceu | Close with solid fill | Close with solid fill | Close with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| Probleme de facultate | Close with solid fill | Close with solid fill | Close with solid fill | Close with solid fill | Checkmark with solid fill |

Tabel 3.1

# Analiză și Fundamentare Teoretică

## Diagrama conceptuala

Diagrama conceptuala a NUME este prezentata in figura 4.1. Această aplicație este împărțită in 3 module principale, si anume:

1. Componenta de front-end a aplicației
2. Componenta de procesare a soluțiilor
3. Componenta API

Diagram

Description automatically generated

Figură 4.1

### Componenta front-end

Componenta de front-end reprezintă modul de interacțiune al utilizatorului cu aplicația. Acesta comunica cu componenta API prin intermediul cererilor de tip HTTP.

### Componenta API

Componenta API este cea responsabila cu tratarea evenimentelor cauzate de interacțiunea dintre utilizator si aplicație. Rolul acestei componente este de a stoca date in baza de date sau încărcarea soluțiilor pe serviciul de stocare Amazon S3.Acesta se ocupa si de funcționalitatea de register si login, si funcționează ca un producer pentru coada de mesaje. Pentru a evita așteptarea si blocarea aplicației in urma trimiterii unei soluții ce necesita mai mult timp de procesare, am decis sa utilizez o coada de mesaje pentru o comunicare asincrona.

### Componenta de procesare a soluțiilor

Componenta de procesare a soluțiilor se ocupa de preluarea soluțiilor si a fișierelor de teste, respectiv referință, de pe serviciul de stocare in cloud Amazon S3. Acesta reprezintă consumerul de pe coada de mesaje, de unde primește ca mesaj id-ul soluției, mai departe rămânând sa isi preia toate datele necesare din serviciul de stocare si din baza de date PostgreSQL.

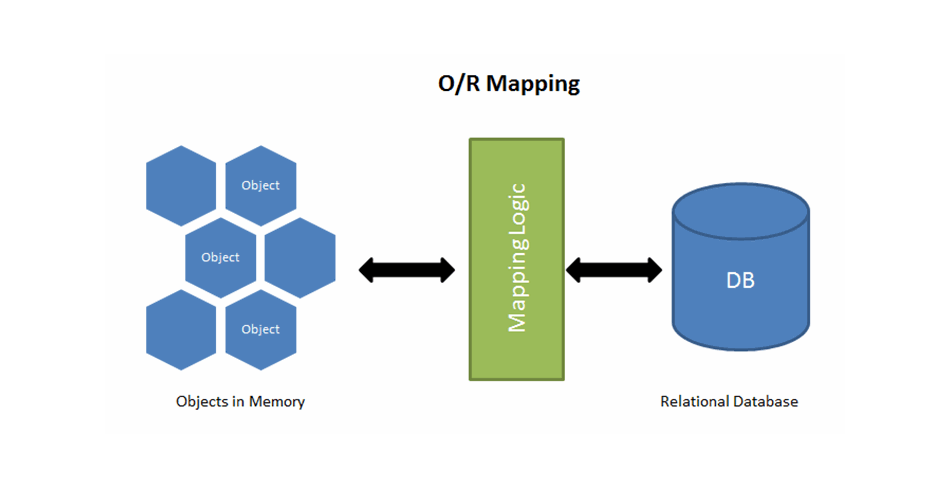
## Tehnologii utilizate

### **PostgreSQL**

PosgreSQL reprezintă un sistem de management(DBMS) object-relational (ORM), open-source, pentru managementul bazelor de date. Acesta poate fi rulat pe toate sistemele de operare populare (Ex: Linux, Windows, Unix etc), suporta date de tip text, imagine, audio, video si permite suport pentru majoritatea limbajelor de programare cum ar fi: Java, Python, C, Go etc.

**ORM** un orm (object-relational mapping) este o procedura prin care se poate face conversia de la obiecte folosite in limbaje de programare bazate pe obiecte cum ar fi Java, in intrari pentru baze de date relaționale. Exemple de astfel de ORM-uri folosite in NUME sunt:

* Hibernate – Hibernate este un ORM produs de Red Hat, pentru limbajul de programare Java care realizează translația dintre clase Java in tipuri de date de tip SQL. Acesta vine la pachet si cu metode predefinite de manipulat date mai ușor, însă permite si crearea unor metode personalizate[?].
* GORM – GORM este un ORM pentru limbajul de programare Go, al cărui rol este de a face translația de la structuri de date, numite modele, in tipuri de date de tip SQL. Acesta vine cu metode predefinite cum ar fi: Save, Find, Delete etc, însă pentru acest ORM este nevoie de mai multa implicare din partea programatorului pentru crearea metodelor personalizate[?].



Figură 4.2

### **Limbaj OOP**

Un limbaj OOP (object-oriented programming) este un model de programare ce organizează datele prelucrate in jurul unor obiecte, in loc de a fi organizat in jurul unor funcții. Obiectele reprezintă o instanță a unei clase, clasa ce este o colecție de atribute si metode specifice acelei clase. NUME este scris in jurul acestui model OOP, mai exact in doua limbaje de programare de tip OOP:

* Java – Java este un limbaj de programare OOP, lansat de Sun Microsystems in 1995. Acesta este un limbaj de programare interpretat, ceea ce înseamnă ca trebuie mai întâi este executat de o mașină virtuala JVM. Serviciul API este realizat in acest limbaj de programare.
* Go – Go reprezintă un limbaj de programare open-source, lansat in 2009 de Google. Go este bazat pe limbajul de programare C, si este un limbaj compilat, ceea ce înseamnă ca orice cod compilat este automat transformat intr-un format înțeles de procesor. Acesta a fost create pentru a simplifica munca programatorului, fiind un limbaj apropiat de C care reprezintă fundația programării. Golang are pe deasupra o funcționalitate numita Goroutines, ce fac limbajul complet concurent. Toate acestea fac limbajul Golang un limbaj perfect pentru servicii de dimensiuni mici. Serviciul de procesare al soluțiilor este realizat in acest limbaj de programare.

### **RabbitMQ**

RabbitMQ este un broker de mesaje open-source, create de Rabbit Technologies Ltd in anul 2007[?]. Aceasta tehnologie permite aplicațiilor participante la comunicare sa se ocupe de partea computaționala, in loc sa consume timp si resurse pe comunicare.

Un **broker de mesaje** este un modul intermediar ce permite comunicarea intre aplicații prin transmiterea de mesaje. Acesta poate fi folosit pentru a reduce timpul de așteptare pe care o cerere de tip HTTP de exemplu l-ar aștepta pentru a primi un răspuns. Astfel, într-o comunicare bazata pe mesaje exista doua tipuri de aplicații participante[?]:

* **Producer** – Producer-ul este aplicația ce produce mesajele si le trimite către un serviciu de management al mesajelor (broker de mesaje). Acest serviciu de management va putea traduce mesajul într-un format înțeles de el si pe care poate sa îl stocheze.
* **Consumer** – Consumerul este receptorul comunicării in acest context. Acesta așteaptă sa apară un mesaj pe coada, respectiv topic in funcție de tipul de comunicare prin mesaje pe care îl avem, pe care sa îl consumer si sa îl utilizeze.

Exista doua tipuri de comunicare bazate pe un broker de mesaje:

* **Publish – Subscribe** – Intr-o comunicare de tipul publish-subscribe, producerul trimite mesaje către un topic. Consumatorii in acest model de comunicare devin abonați la topicul pe care trimite producerul mesaje. In acest tip de comunicare pot trimite mai mulți produceri mesaje pe un topic, iar la un singur topic se pot abona mai mulți abonați(Figura 4.3).
* **Point-to-point** – Intr-o comunicare de tipul point-to-point, se folosește o coada pentru a stoca mesajele. Fiecare coada poate primi mesaje de la mai mulți produceri, si mai mulți consumer pot primi mesaje stocate pe aceeași coada. Producerul trimite mesajele către coada, pentru ca mai târziu acestea sa fie consumate de către producer si trimite un mesaj de confirmare către broker, pentru a asigura ca mesajul a fost consumat cu succes(Figura 4.4). Acest tip permite o comunicare asincrona pentru servicii.

Graphical user interface, application

Description automatically generated

Figură 4.3

Diagram

Description automatically generated

Figură 4.4

RabbitMQ folosește o un model de comunicare derivat de la modelul point-to-point, deoarece acesta nu leagă direct pdoucer-ul la coada. Fiecare producer va trimite mesaje către un „exchange”. Acest exchange are rolul de a se asigura ca fiecare mesaj ajunge in coada corecta, folosindu-se de alte doua elemente numite: routing key si binding. Binding-ul reprezintă legătura dintre o coada si un exchange, legătură care este realizata cu ajutorul unui routing-key, asemănător unei adrese[?].

Graphical user interface, diagram

Description automatically generated

Figură 4.5

Pentru serverul de RabbitMQ am ales sa folosesc un serviciu de RabbitMQ oferit de CloudAMQP, care este responsabil de managementul acestei tehnologii.

### **Amazon S3**

Amazon S3 este un serviciu de stocare in cloud, oferit de către Amazon Web Services, bazat pe obiecte (datele stocate) si AWS buckets (containere de date). Fiecare obiect este apoi găsit printr-o adresa, construita folosind numele obiectului si al bucketului[?].

Graphical user interface, application

Description automatically generated

Figură 4.6

Acest serviciu de stocare a venit ca o soluție pentru companii la problema costului achiziționarea resurselor, menite doar pentru stocarea informațiilor. Astfel prin Amazon S3, companiile pot achiziționa la un preț mic mai mult spațiu de stocare in cloud, oricând este nevoie.

**Cloud** reprezintă o tehnologie apărută cu creșterea exponențială a nevoii de resurse noi. Aceasta tehnologie pune la dispoziție servicii online, accesate prin intermediul internetului, fără a mai fi nevoie sa rulam aceste servicii pe un calculator fizic. Prin ajutorul cloudului, utilizatorii pot accesa aceleași fișiere de pe orice dispozitiv, datorita faptului ca ele sunt rulate pe servere in centre de date, in loc de a fi rulate pe mașini locale, atât timp cat are credentialele corespondente. Aceasta tehnologie este posibila cu ajutorul virtualizării[?].

Diagram

Description automatically generated

Figură 4.7

**Virtualizarea** este tehnologia care permite crearea unor calculatoare virtuale, care funcționează ca niște calculatoare fizice, numele lor fiind „Mașini Virtuale”. Aceste mașini virtuale își preiau resursele din calculator fizic pe care sunt rulate, însă nu își afectează aplicații sau fișiere intre ele, astfel probleme apărute pe o mașină virtuala nu afectează si mașina fizica. Un server reprezintă un calculator fizic, ce menține mai multe mașini virtuale pornite in același timp. Furnizorii de servicii de cloud, cum ar fi Amazon Web Services(AWS), Microsoft Azure, Google Cloud etc, își rulează serviciile pe mai multe calculatoare si in regiuni diferite, evitând probleme cauzate de defectele unui singur server. Astfel serviciile oferite sunt mereu online si pregătite sa fie accesate.

### **Spring Boot**

Spring boot reprezintă un framework de Java open-source, care permite transformarea arhitecturilor monolit in arhitectura bazata pe micro servicii.

**Arhitectura bazata pe micro servicii** este un model arhitectural ce permite programatorilor sa creeze aplicații de dimensiuni mici, menite sa realizeze funcționalități legate intre ele din punct de vedere comportamental. Aceasta arhitectura vine la rândul ei cu un set de avantaje:

* Sistemele permit integrarea unor noi funcționalități, nefiind necesar sa modificam aplicația de baza.
* Componentele pot fi organizate in containere.
* Aplicațiile au nevoie de mai puțin timp pentru a fi implementate, ele putând fi create incremental.

Diagram

Description automatically generated

Figură 4.8

### **Docker**

Docker este o platforma ce permite împachetarea aplicațiilor in medii izolate, numite containere. Aceste containere se aseamănă cu mașini virtuale, prin prisma faptului ca acestea nu au nevoie de dependințe externe din partea mașinii fizice pe care sunt rulate. Odată cu crearea acestor containere, cu ajutorul unui Dockerfile, ne asiguram ca acel container va avea toate dependințele necesare aplicației, astfel aplicația va rula fără probleme pe orice calculator fizic rulam containerul respectiv. Containerele docker se diferențiază totuși de mașini virtuale, prin faptul ca acestea nu au propriul sistem de operare si propriile resurse, ele folosindu-se de cele deja existente pe calculatorul host.[?]

**Dockerfile** reprezintă un fișier text care conține toate comenzile necesare pentru a crea o imagine. Aceasta imagine poate fi folosita pentru a reproduce din nimic mediul in care a fost rulata aplicația printr-o simpla comanda numita „Docker build”.[?]

**Docker Swarm** reprezintă o unealta de orchestrare al containerelor docker. Prin folosirea acestei tehnologii, cream mai multe noduri intr-un cluster, fiecare nod fiind capabil sa acceseze si sa susțină orice container. Docker Swarm ne permite sa asiguram fiabilitatea programului, acesta venind cu o funcționalitate de back-up in cazul întâlnirii erorilor. Docker Swarm poate fi considerat astfel încât sa permită restartarea containerelor atunci când acestea se închid din cauza unei probleme fatale. [?]

Graphical user interface, table

Description automatically generated

Figură 4.9

### **React**

React reprezintă o librărie de JavaScript, menita sa faciliteze implementarea interfețelor utilizator. React introduce conceptul de componente, care permite construirea unor componente mai mici, si apoi combinarea si reutilizarea acestora pentru a construi interfețe utilizator complexe. React este un limbaj de tip declarativ. Acesta este însărcinat sa preia codul si sa îl transforme într-o pagina HTML care sa respecte codul scris de programator.

## Protocoale utilizate

### **HTTP**

Protocolul HTTP reprezintă fundația comunicării într-o arhitectura de tip client-server. Modul in care funcționează comunicarea HTTP este prin intermediul unei conexiuni sincrone, in care clientul trimite o cerere către server, apoi așteaptă ca acesta sa ii răspundă cu un mesaj de confirmare. Tipurile de cereri ce pot fi făcute prin protocolul HTTP sunt:

* GET – cererile de tip GET sunt folosite pentru a cere serverului informații despre datele sale.
* POST – cererile de tip POST sunt folosite pentru a trimite date catre server. Aceste cereri sunt însoțite de un conținut.
* OPTIONS – cererile de tip OPTIONS sunt folosite pentru a afisa tipurile de metode HTTP ce pot fi folosite pe un anumit URL.
* DELETE – cererile de tip DELETE sunt folosite pentru a-i cere serverului sa șteargă resurse.
* TRACE – cererile de tip TRACE sunt folosite pentru a accesa mesajele trimise pe un anumit URL

### **S3**

Protocolul s3 este un protocol utilizat pentru a putea accesa locația obiectelor aflate in serviciul de stocare Amazon S3. Acest protocol permite o stocare durabila, securizata si scalabila a obiectelor[?].

### **AMQP**

Protocolul AMQP (Advanced Message Queuing Procol) este un protocol ce permite comunicarea prin intermediul mesajelor intre aplicații. Acesta asigura o comunicare sigura si fiabila[?].

## Decizii de implementare

* **Folosirea unei comunicări bazate pe cozi de mesaje:** Prin folosirea bazata pe cozi de mesaje asiguram o comunicare asincronă intre cele doua servicii ce împreună compun back-end-ul aplicației. Soluțiile problemelor pot sa difere din punct de vedere al timpului de rulare, astfel utilizatorul ar întâmpina multe momente de downtime in care aplicația ar rămâne blocata așteptând rezolvarea unei soluții. Acest blocaj ar putea fi cauzat de utilizatorul in cauză, sau chiar de un alt utilizator ce folosește aplicația in acest timp. Aceasta decizie a fost luata pentru a asigura cerințele non-funcționale de scalabilitate si performantă. Prin folosirea unei cozi de mesaje, utilizatorul poate transmite o soluție si apoi sa isi continue navigarea pe platforma.
* **Folosirea unui serviciu de stocare cloud:** Prin folosirea unui serviciu de stocare cloud (Amazon S3), asiguram scalabilitatea aplicației. ???????????????????????????????????????????????????
* **Kafka vs RabbitMQ:** RabbitMQ reprezintă un serviciu de comunicare bazat pe mesaje, al carui rol principal este comunicarea sub forma de cerere-răspuns. RabbitMQ este mai strict decât Kafka din punct de vedere al păstrării ordinii mesajelor[?].
* **Împărțirea aplicației in micro servicii:** Pentru o testare si extindere mai ușoară a sistemului am ales sa folosesc o arhitectura bazata pe micro servicii in loc de o arhitectura monolit. Împărțirea pe micro servicii face si înțelegerea funcționalităților mai ușoară, împărțind componentele pe baza comportamentului lor:
  + serviciul API este responsabil doar de tratarea cererilor făcute de client si asignarea responsabilităților către micro serviciile corespondente acțiunii făcute de utilizator.
  + serviciul de procesare al soluțiilor se ocupa doar de procesarea compilarea si rularea soluțiilor.

Testarea celor doua componente poate fi făcută separat, urmărind mai ușor fluxul cazului de utilizare.

Scalabilitatea fiecărui serviciu poate fi realizata independent, fără a afecta alte componente.

* **Folosirea limbajului Golang:** Pentru serviciul de procesare al soluțiilor am ales sa folosesc Go deoarece acesta este unul de dimensiuni mici, așa ca nu are nevoie de funcționalități complexe puse la dispoziție de limbaje precum Java. Acesta fiind un limbaj compilat, este mai rapid decât un limbaj interpretat, ceea ce ne dorim de la un serviciu de procesare.

# Proiectare de Detaliu si Implementare

## Arhitectura sistemului

Diagram, schematic

Description automatically generated

Figură 5.1

Pentru implementarea NUME, au fost folosita o arhitectura de tip micro servicii. Prin folosirea acestui model arhitectural, aplicația a fost împărțită in mai multe componente independente, fiecare componenta având responsabilitatea sa, comunicarea dintre componente fiind realizata in diferite moduri.

### **Arhitectura API – Front-end**

Pentru implementarea legaturii dintre serviciul API si front-end-ul aplicatiei, am ales o arhitectura de tip client-server(Figura 5.2).

Diagram

Description automatically generated

Figură 5.2

In aceasta arhitectura de tip client-server, clientul reprezinta o interfata pusa la dispozitie utilizatorului pentru a putea realiza functionalitatile aplicatiei, iar serverul reprezinta componenta ce are rolul de a consuma datele primite de la client si a realiza toate sarcinile computationale. Comunicarea dintre cele doua componente se face prin intermediul unor cereri REST.

### **Implementare serviciu API**

**Diagrama de pachete**

**Diagram

Description automatically generated**

Figură 5.3

Serviciul API a fost implementat folosind o arhitectura bazata pe nivele[?]. Pachetele de clase sunt impartite in patru nivele, organizate vertical, comunicarea intre aceste nivele fiind realizabila doar intre nivelele vecine. Prin aceasta izolare a nivelelor, se realizează o impartire logica a codului, astfel putem fi siguri ca modificarile dintr-un nivel, nu vor aduce probleme neasteptate in alte nivele ale aplicatiei. Cele patru nivele ale componentei sunt:

* Presentation Layer = Acesta este nivelul care se ocupă de interfata utilizator si tratarea evenimentelor cauzate de interacțiunea cu aplicatia. Deoarece am decis ca interfața utilizator sa fie realizată într-o componenta separata, acest membru al nivelului va lipsi. Pachetele ce fac parte din Presentation Layer sunt:
  + Controllers – Acest pachet contine clasele ce se ocupa de acceptarea cererilor REST venite din partea componentei front-end. In urma unei cereri, controller-ul are rolul de a redirectiona responsabilitatea de a indeplini sarcina catre componenta corecta
* Business Layer = Acesta este nivelul care se ocupa de procesarea informatiilor si de realizarea functionalitatilor. Clasele din pachetul controller, vor redirectiona responsabilitatea catre clase din acest nivel. Pachetele ce fac parte din Business Layer sunt:
  + Dtos – dtos sau Data Transfer Objects sunt clase menite pentru a prelucra doar informatiile necesare. Prin folosirea dtos putem trimite catre interfata utilizator doar informatiile necesare, ascunzand astfel posibile date nedorite.
  + Builders – Clasele din pachetul builder sunt clase menite pentru a face conversia de la entitati la dtos.
  + Configs – Clasele din pachetul configs sunt clase ce se ocupa de configurari precum ar fi securizarea aplicatiei, conexiunea la serviciul de stocare sau crearea legaturii cu coada de mesaje RabbitMQ
  + Services – Clasele din pachetul services sunt clasele ce contin logica din spatele implementarii tuturor functionalitatilor aplicatiei. Aceste clase comunica direct cu pachetul Controllers.
  + Exceptions – In pachetul exceptions se afla clase create pentru personalizarea erorilor ce pot aparea in timpul executiei functionalitatilor.
  + Security – In pachetul security, se afla clasele ce asigura securitatea utilizarii aplicatiei. In aceste clase se realizeaza autentificarea prin protocolul oAuth sau generarea tokenului JWT.
  + Validators – In pachetul validators exista clase menite validarii datelor aplicatiei.
  + Utils – Pachetul utils contine diferite functionalitati ce sunt necesare mai multor componente, dar nu au putut fi integrate in pachetul services deoarece acestea nu erau legate din punct de vedere logic. In acest pachet avem functionalitati precum incarcarea fisierelor pe S3 sau returnarea caii catre un fisier.
* Persistence Layer = Acesta este nivelul care se ocupa de toate operatiile si datele legate de baza de date. Pachetele ce se afla in Persistence Layer sunt:
  + Entities = Pachetul entities contine clase echivalente tabelelelor din baza de date.
  + Repositories = Clasele din pachetul repositories au rolul de a interactiona direct cu baza de date. ORM-ul hibernate vine cu un set de metode predefinite pentru a realiza aceasta interactiune.

Diagram

Description automatically generated

Figură 5.4

### **Implementare serviciul de procesare**

**Diagrama de pachete**

**Diagram, schematic

Description automatically generated**

Figură 5.5

Pentru serviciul de procesare, nu am folosit o anumita arhitectura, aceasta avand o structura standard monolith. In diagrama de pachete(Figura 5.5) se poate observa relatia dintre cele 5 pachete ale aplicatiei:

* Repository – In pachetul repository se afla toate clasele ce au rolul de a accesa date dintr-o sursa externa, mai exact din fisierele locale ale calculatorului, din serviciul de stocare in cloud sau din baza de date.
* Entities – Pachetul entities contine structurile folosite in componenta, simuland obiectele folosite itnr-un limbaj OOP. Aici intalnim obiecte precum configurari, structuri echivalente tabelelor din baza de date
* Services – Pachetul services contine clasele ce tin de functionalitati speciale ale aplicatiei, ce nu pot fi integrate in pachetul language\_runners, cum ar fi consumerul de pe coada de mesaje.
* Runners – Pachetul ce contine logica computationala a componentei. Acest pachet este impartit in alte doua subpachete si anume:
  + Executions = In pachetul executions intampinam fisiere ce au rolul de a executa comenzile de compilare si rulare ale solutiei. Aici gasim fisiere precum „executor\_service” al carui singur rol este de a executa solutia si a returna rezultatul acestei executii, dar si fisierul memory\_monitor al carui rol este da calcula memoria utilizata de executie.
  + Language\_runners = Pachetul language runner este compus din clase al caror scop este da rula solutia trimisa de utilizator in limbajul de programare ales de acesta. Aici intalnim fisierele specifice fiecarui limbaj de programare, dar si un fisier numit „submission\_wrapper”, al carui rol este de a coordona fluxul corectarii unei solutii si de a asigna responsabilitatea fisierului de rulare corespunzator, in functie de limbajul de programare.
* Custom\_errors – In acest pachet avem cele doua tipuri de erori ce pot aparea in timpul executiei unei solutii trimise de catre utilizator:
  + TLE = Time limit exceeded, atunci cand executia solutiei dureaza mai mult decat timpul limita pe care il permite problema
  + MLE = Memory limit exceeded, atunci cand executia solutiei dureaza mai mult decat memoria limita pe care o permite problema.

## Structura bazei de date

Pentru acest proiect am ales PostgreSQL pentru a implementa o baza de date rațională. Comunicarea dintre cele doua servicii au fost realizate folosind cele doua ORM prezentate in capitolul anterior: Hibernate si GORM. Pentru baza de date am ales o baza de date păstrată in cloud pusa la dispoziție de Heroku.

Conectarea la baza de date a fost realizata prin menționarea unui URL specific in fișierul *application.properties* din proiectul serviciului API, respectiv prin adăugarea aceluiași URL in apelul metodei de inițializare a bazei de date, din serviciul de procesare al soluțiilor.

Diagram

Description automatically generated

Figură 5.6

In figura 5.6 se poate observa structura bazei de date, prezentând tabelele si relațiile dintre ele. Descrierea fiecărei componente fiind următoarea:

* **Users:** Tabela *users* păstrează toata datele legate de utilizatorii aplicatiei. Acest tabel este format din:
  + Un id unic
  + Numele pe care utilizatorul dorește sa îl afișeze in aplicație
  + Adresa email a utilizatorului
  + Parola utilizatorului, codificata
  + Un id unic si numele furnizorului, in cazul in care utilizatorul decide sa se înregistreze folosind un cont creat printr-un serviciu extern.
* **Roles:** Tabelul *roles* are rolul de a păstra rolurile pe care le are un utilizator. Acestea pot fi Admin sau Utilizator. Aceasta tabela conține:
  + Un id unic
  + Numele rolului.
* **User\_Role:** Tabelul *user\_role* are rolul de a crea o relație many-to-many intre roluri si utilizatori. Fiecare utilizator poate avea mai multe roluri, dar si un rol poate fi asignat mai multor utilizatori.
* **Submission:** Tabelul *submissions* are rolul de a păstra toate datele legate de soluțiile trimise de utilizatori. Acest tabel este format din:
  + Un id unic
  + Un id al problemei pentru care a fost trimisa soluția
  + Un id al utilizatorului care a trimis soluția (împreună cu coloana precedenta creează o relație de tip many-to-one cu tablele precizat).
  + Limbajul de programare in care a fost scrisa solutia
  + Data si timpul la care a fost încărcată soluția
  + Statusul soluției, care poate lua valorile: FAILED, IN PROGRESS sau SUCCESS.
* **Test\_results:** Tabelul *test\_results* are rolul de a păstra toate datele legate de rezultatele testelor. Acest tabel este format din:
  + Un id unic
  + Id-ul solutiei pentru care înregistrează rezultatul testului, creând o relație de tip many-to-one
  + Statusul testului, care poate fi adevărat daca rezultatul rulării este același cu cel din fișierul de referință.
  + Timpul necesar pentru a se termina rularea soluției
  + Memoria utilizata prin alocarea de resurse
  + Un mesaj de eroare in cazul in care soluția a întâmpinat o eroare in timp ce rula testul.
* **Problems:** Tabelul *problems* are rolul de a păstra toate datele legate de problemele existente pe NUME:
  + Un id unic
  + Dificultate problemei. Aceasta poate lua valorile: EASY, MEDIUM sau HARD
  + Un exemplu de date de intrare
  + Un exemplu de date de ieșire
  + Enunțul problemei
  + Titlul problemei
  + Limita de timp in care soluția trebuie sa se încadreze
  + Limita de memorie in care soluția trebuie sa se încadreze
* **Test\_cases:** Tabelul *test\_cases* are rolul de a păstra date legate de setul de teste pe care sa fie rulata soluția:
  + O coloana care păstrează numele fișierului ce conține datele de intrare specifice testului
  + O coloana care păstrează numele fișierului ce conține datele de ieșire specifice testului
  + Id-ul problemei de care testul aparține, creând astfel o relație de tipul many-to-one cu tabelul *problems*

## Fluxurile aplicației

### **Înregistrare si autentificare**

Autentificarea in aplicație este primul pas pe care trebuie sa îl facă orice utilizator pentru a putea realiza orice alta acțiune. Dar in primul rând pentru a se putea autentifica, un cont personal al utilizatorului trebuie sa existe. Acest lucru se face posibil prin acțiunea de înregistrare(Figura 5.3).

**Pentru a se putea înregistra**, utilizatorul va trimite datele noului sau cont către server prin intermediul clientului. Aceasta cerere va fi una de tip POST si trimisa pe endpointul „api/auth/signup”. Serverul verifica daca adresa de email a utilizatorului se afla deja in baza de date. In caz contrar, serverul va salva noul utilizator in baza de date si va returna un mesaj de confirmare a înregistrării.

Diagram

Description automatically generated

Figură 5.7

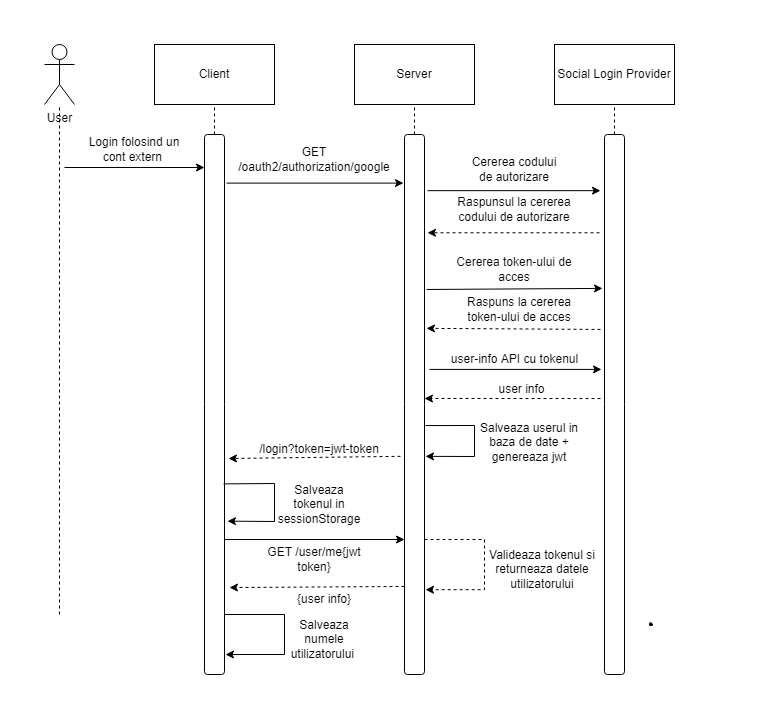
**Operația de autentificare** (Figura 5.4) se face prin folosirea unui token JWT(JSON Web Token). Utilizatorul își introduce adresa de email si parola, care mai apoi sunt trimise de către client printr-o cerere de tipul POST către server. Serverul validează ca într-adevăr credențialele exista in baza de date si utilizatorul este înregistrat. Daca utilizatorul nu exista, serverul va returna un mesaj de eroare. In caz contrar, serverul va returna un token împreună cu datele utilizatorului, rămânând ca clientul sa le salveze in sessionStorage token-ul si numele utilizatorului. SessionStorage este un mecanism de salvare a datelor pentru sesiunea curenta. Odată cu închiderea ferestrei, datele din sessionStorage vor fi șterse, spre deosebire de localStorage unde datele persista. Pentru a putea realiza operațiile aplicației, clientul va adăuga acum ca header de autentificare token-ul primit de la server.

Diagram

Description automatically generated

Figură 5.8

Daca utilizatorul nu dorește sa treacă prin cele doua procese prezentate mai sus, acesta se poate autentifica oricând folosind un cont extern de Google folosind **protocolul OAuth**. Acest protocol se folosește de un token de autorizare pentru a evita expunerea datelor confidențiale cum ar fi parola, aplicației ce consuma acest cont. Inițial se face o cerere pentru un cod de autorizare catre furnizor, google in cazul nostru. Cu acest cod primit, serverul va face o alta cerere cu scopul de a primi un token generat de furnizor, pentru ca apoi sa preia datele utilizatorului folosind acest token. Serverul va salva aceste date, in cazul in care utilizatorul nu exista deja si va returna un token JWT care va fi salvat de client. La final clientul trimite o cerere de tip GET pentru a primi datele utilizatorului, folosind tokenul JWT si salvează datele necesare.



Figură 5.9

### **Adăugarea unei probleme**

Pentru adăugarea unei probleme, utilizatorul trebuie sa fie in primul rând logat in aplicație. Acesta accesează pagina cu probleme, clientul trimițând o cerere de GET către server. După ce serverul returnează toate problemele si pagina ii este afișată utilizatorului, acesta va apăsa pictograma ce sugerează adăugarea unei probleme, lucru ce va începe acest flux de execuție. Clientul afișează pagina de introdus datele noii probleme, unde utilizatorul va completa câmpurile si selecta fișierele de teste pe care dorește sa le încarce, fiecare fișier de date de intrare având un corespondent un fișier de date de ieșire si viceversa. La final, acesta apasă butonul de „Save”, trimițând o cerere de tip POST către server, conținând datele noii probleme. Fișierele de teste vor fi trimise către serviciul de stocare Amazon S3, unde vor fi salvate sub un director cu numele „Problems/*NumeleProblemei/*inputs*/” sau „*Problems*/NumeleProblemei/*references/”. Celelalte date ale problemei vor fi salvate in baza de date, iar la final serverul va trimite un mesaj de confirmare sau de eroare, în funcție de rezultatul pe care îl are operația de salvare.

Diagram

Description automatically generated

Figură 5.10

### **Vizualizarea unei soluții**

In aplicația NUME utilizatorul își poate vede soluțiile trimise anterior. Înainte de a începe aceasta actiune, utilizataorul trebuie sa fie logat in aplicație. Acesta accesează pagina cu soluții, clientul trimițând o cerere de GET către server. După ce serverul returnează toate soluțiile si pagina ii este afișată utilizatorului, acesta poate selecta o soluție apăsând butonul de „View”, prezent pe fiecare rând al tabelului de soluții. Clientul trimite o cerere de tip GET către server, apoi serverul preia textul soluției de pe serviciul de stocare in cloud, iar datele soluției din baza de date. La final serverul returnează toate datele soluției, iar clientul afișează o pagina populata de aceste date. Utilizatorul poate vizualiza date despre fiecare test in parte, accesând elementele unei liste sub forma de acordeon.

Diagram

Description automatically generated

Figură 5.11

### **Rezolvarea unei probleme**

Rezolvarea unei probleme este funcționalitatea principala a aplicației. Înainte de a începe aceasta acțiune, utilizatorul trebuie sa fie logat in aplicație. Trimiterea unei solutii incepe prin accesarea paginii de probleme. Clientul va face o cerere de tip GET către server pe endpointul „*problems/all*”, apoi serverul va returna o lista cu toate problemele existente. Utilizatorul isi va putea alege una din probleme, apăsând pe butonul „Try”, asociat fiecărei linii din tabelul de probleme. Prin apăsarea butonului, clientul va declanșa o cerere de tip GET către server folosind endpointul problem/{titlul problemei}, la care acesta va primi detaliile problemei cerute. Clientul ii va afișa utilizatorului pagina problemei, pagina pe care acesta trebuie sa completeze textul soluției. Utilizatorul va decide sa trimită soluția către corectat, clientul trimițând o cerere de tip POST către server pe endpointul „submissions/add”. Serverul va salva textul soluției pe serviciul de stocare in cloud Amazon S3, iar datele soluției in baza de date. Daca aceste doua operații au fost îndeplinite cu succes, id-ul soluției va fi salvat într-o coada de mesaje.

Serviciul de procesare al soluțiilor preia soluții din coada de mesaje de fiecare data când este disponibil. Când un mesaj nou a fost consumat, acest serviciu începe fluxul de rulare al soluției. Rularea soluției începe prin preluarea textului acesteia si a fișierelor de teste de pe serviciul de stocare in cloud, dar si datele soluției si problema pentru care a fost propusă această soluție din baza de date. Acest serviciu rulează codul trimis de către utilizator, iar după finalizarea acestei rulări salvează rezultatele testelor si actualizează statusul soluției.

Diagram, schematic

Description automatically generated

Figură 5.12

## Structura organizării fișierelor in S3

Diagram

Description automatically generated

Figură 5.13

Pentru a avea o structură logica a fișierelor, organizarea acestora a fost realizată ca in figura 5.13. Fiecare problemă nouă va fi salvată in fișierul cu numele său, din directorul „problems”, iar fiecare soluție nouă va fi salvată sub fișierul cu numele problemei pentru care a fost trimisă, din directorul „submissions”.

## Detalii de implementare

### **Comunicarea intre serviciul API si cel de procesare a soluțiilor**

Comunicarea intre cele doua servicii se va face printr-o coada de mesaje pusa la dispoziție de RabbitMQ. Atunci când o soluție este trimisa spre corectare, serviciul API va trimite pe coada de mesaje un mesaj ce contine id-ul soluției. RabbitMQ pune la dispoziție o coada de mesaje ce poate accepta mai multi consumatori. In cazul NUME

serviciul API reprezintă producerul, iar serviciul de procesare reprezintă consumerul acestei cozi. Deoarece aceste doua componente sunt împărțite in doua module independente, singura legătură dintre ele fiind aceasta coada, putem instanția mai multe servicii de procesare.

RabbitMQ reprezintă o coada de tip FIFO (First in, first out), însemnând ca păstrează ordinea mesajelor, in funcție de cum au ajuns in coada. Modul in care RabbitMQ tratează mesajele consumate este așteptarea unui mesaj de confirmare ca mesajul a fost consumat cu succes. In cazul care acest răspuns nu a fost primit, coada va păstra mesajul pana acesta va fi consumat in mod corespunzător. Prin acest mesaj de confirmare, RabbitMQ păstrează persistenta mesajelor.

### **Stocarea fișierelor in Amzaon S3**

Pentru a salva textul soluțiilor, problemelor si fișierele de teste, este folosit serviciul de stocare in cloud Amazon S3. Aceste fișiere vor fi stocate intr-un bucket special „NUME\_bucket\_1”, bucket aflat in regiunea eu-central-1.

Pentru a putea fi folosite metodele de GET si PUT trebuie in primul rând să creăm o sesiune s3. Aceasta ulterior având aceste doua metode. Operațiile de salvare si descărcare vor fi realizate folosind cele doua metode menționate. Fișierele de teste însă vor fi descărcate intr-un mod special in serviciul de procesare al soluțiilor folosind biblioteca Golang **„github.com/seqsense/s3sync”**. Aceasta biblioteca ne pune la dispoziție metoda sync, care primește ca prim parametru calea către fișierul de pe serviciul de stocare, iar ca al doilea parametru primește calea către directorul local. Aceasta metoda se va asigura ca atât directorul local, cat si cel de pe cloud vor avea același conținut.

### **Execuția si evaluarea soluțiilor**

# Testare și Validare

# Manual de Instalare si Utilizare

## Instalarea sistemului

## Utilizarea aplicației

Un utilizator normal al aplicației nu are nevoie sa urmeze pașii de instalare ai sistemului. In momentul in care aplicația este activa, in urma instalării acesteia de către un administrator, utilizatorul normal poate începe sa o folosească.

### **Înregistrare si autentificare**

Graphical user interface, application

Description automatically generated

Figură 7.1

In urma accesării platformei, utilizatorul este întâmpinat de pagina de autentificare. Acesta poate accesa aplicația prin introducerea adresei de email si a parolei cu care s-a înregistrat, finalizând acțiunea prin apăsarea butonului de „SIGN IN”. In cazul in care acesta nu are un cont înregistrat in aplicație, el poate opta prin a se autentifica cu una din rețelele de socializare prin pictograma corespunzătoare, sau se poate înregistra in aplicație accesând pagina de „SIGN UP” din antet.

Pentru a se înregistra, utilizatorul își introduce datele in căsuțele din pagina cu care este întâmpinat. In cazul in care utilizatorul dorește sa se autentifice ca un administrator, trebuie sa introduce un cod de validare. La final utilizatorul apasă butonul de „SIGN UP” si va fi redirecționat către pagina de „SIGN IN” pentru a se autentifica cu noul cont.

### **Adăugare problemă**

Graphical user interface, text, application

Description automatically generated

Figură 7.2

Pentru a adăuga o problema noua, utilizatorul accesează pagina de probleme prin apăsarea butonului „Problems” din meniul din stânga al aplicației, primind astfel lista de probleme. Pentru a continua acțiunea, utilizatorul va apăsa butonul reprezentat de pictograma „+”, care îl va duce către pagina de introducere a detaliilor.

Graphical user interface, application

Description automatically generated

Figură 7.3

Utilizatorul va introduce detaliile problemei, adică titlul, enunțul, un exemplu de date de intrare si de ieșire, dificultate, limita de memorie si de timp. Acesta va trebuia apoi sa încarce si fișierele de teste pentru test, pentru ca soluțiile să poată fi rulate pe acest set.

Observații:

* Fișierele trebuie sa fie denumite astfel încât sa respecte formatul „TitluProblemă\_NumeTest.in” sau „TitluProblemă\_NumeTest.ref”
* Toate fișierele ce conțin date de intrare trebuie sa aibă un corespondent in fișiere de ieșire si viceversa.
* Numele problemelor este unic, deci utilizatorul trebuie sa se asigure ca problema nu exista deja.

A picture containing text, screenshot, monitor, indoor

Description automatically generated

Figură 7.4

La final utilizatorul apasă butonul „Save” pentru a salva problema. Daca salvarea problemei nu întâmpină nicio problema, utilizatorul va fi redirecționat către pagina de probleme, in caz contrar acestuia ii va apărea un mesaj de eroare.

### Trimiterea unei soluții

Pentru a încerca sa rezolve o problema, utilizatorul accesează pagina de probleme prin apăsarea butonului „Problems” din meniul din stânga al aplicației, primind astfel lista de probleme.

Graphical user interface, text, application

Description automatically generated

Figură 7.5

Utilizatorul își alege problema pe care dorește sa o încerce apăsând butonul „Try”, prezent pe fiecare rând al tabelului de probleme. Acest buton îl va duce către o pagina unde va vedea detaliile problemelor, dar si un editor de text integrat in browser, pentru a putea rezolva cerința direct in aplicație.

Graphical user interface, text, application

Description automatically generated

Figură 7.6

Pentru a putea reveni ușor la cerința problemei, acesta va putea vizualiza in stânga paginii enunțul acesteia, urmat de un exemplu de date de intrare si unul de date de ieșire, facilitând înțelegerea cerinței. In dreapta ferestrei se va afla un editor de text, inițializat cu un schelet de cod predefinit pentru limbajul de programare ales, pe care poate continua sa își completeze soluția. Apăsând dropdownul de sub editorul de text, utilizatorul poate alege limbajul de programare in care dorește sa rezolve problema, lucru ce va întoarce editorul de text într-o stare inițială specifica pentru alegerea făcută. Cand utilizatorul este sigur ca a introdus soluția corecta, acesta poate apăsa butonul „Submit” fiind redirecționat către pagina de probleme. In cazul unei erori, un mesaj de eroare va apărea pentru a îl atenționa ca soluția nu s-a putut trimite cu succes.

### **Vizualizare soluții anterioare**

Pentru a vizualiza o soluție anterioară, utilizatorul accesează pagina de soluții prin apăsarea butonului „Submissions” din meniul din stânga al aplicației, primind astfel lista de soluțiile personale propuse anterior.

A picture containing graphical user interface

Description automatically generated

Figură 7.7

In aceasta pagina utilizatorul poate vedea toate soluțiile propuse anterior, pentru fiecare primind data si ora la care a trimis soluția, titlul problemei pentru care a trimis soluția si statusul acestei soluții. Fiecare rând va fi colorat cu o culoare specifica statusului soluției:

* Verde: SUCCESS
* Galben: IN PROGRESS
* Roșu: FAILED

Soluțiile IN PROGRESS reprezintă soluții care încă sunt in așteptare pentru a fi evaluate, utilizatorul putând verifica statusul acesteia reîncărcând pagina. Pentru a vizualiza detaliile unei soluții, utilizatorul poate selecta soluția respectivă apăsând butonul „View”, prezent pe fiecare rând al tabelului de soluții.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 7.8

Odată accesată pagina de detalii ale soluției, utilizatorul poate vedea problema pentru care a fost trimisa soluția si enunțul acesteia. In cazul in care el dorește sa acceseze rapid problema, pentru a încerca sa o rezolve, poate apăsa pe titlul acesteia. Pe aceasta pagina mai sunt prezente si detalii precum soluția propriu zisa, timpul si data in care a fost încărcată soluția, limita de memorie si de timp, limbajul de programare in care a fost trimisa soluția si statusul acesteia. Pentru ca utilizatorul sa poată vedea rezultatul pentru un anumit test, acesta poate selecta testul respectiv printr-un click pe numele testului. Detaliile rezultatului se vor extinde pentru a vizualiza timpul de execuție, memoria folosită si statusul testului respectiv.

Graphical user interface

Description automatically generated

Figură 7.9

# Concluzii

Acest capitol va ocupa circa 5% din lucrare.

Capitolul ar trebui sa conțină (nu se rezumă neapărat la):

* un rezumat al contribuțiilor voastre
* o analiză critică a rezultatelor obținute
* o descriere a posibilelor dezvoltări și îmbunătățiri ulterioare

# Bibliography

|  |  |
| --- | --- |
| [1] | G. Cachon și M. Lariviere, „Supply chain coordination with revenue sharing contracts: strengths and limitations,” *Management Science,* vol. 51, pp. 30-44, 2005. |
| [2] | G. Boella și L. van der Torre, „Contracts as Legal Institutions in Organizations of Autonomous Agents,” în *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*, New, 2004. |
| [3] | G. Boella, J. Hulstijn și L. van der Torre, „A Synthesis Between Mental Attitudes and Social Commitments in Agent Communication Languages,” în *Intelligent Agent Technology 05 (IAT 2005)*, Compiegne, 2005. |
| [4] | Software Freedom Conservancy, „The Selenium Browser Automation Project,” [Interactiv]. Available: https://www.selenium.dev/. [Accesat 17 martie 2021]. |
| [5] | C. P. Pfleeger, S. L. Pfleeger și J. Margulies, Security in Computing, 5th Edition, Pearson, 2015. |

După ce ați inserat/actualizat bibliografia selectați întregul tabel și aplicați stilul *Biblio.*  Stilul *Normal* are indentare la începutul paragrafelor și, de aceea nu veți obține formatul ca mai sus fără acest pas.

# Anexa 1

…

Secțiuni relevante din cod

…

Alte informații relevante (demonstrații etc.)

…

Lucrări publicate (dacă există)

etc.