1. INTRODUCERE

1.1 DESCRIEREA PROBLEMEI

În ziua de astăzi, devine din ce în ce mai greu pentru proaspeții absolvenți de studii superioare să își găsească un loc de muncă. Acest lucru se datorează în principal numărului mare de absolvenți și a numărului de locuri de muncă mult prea mic pentru a satisface cererea. Această problemă a devenit prezentă în mai toate sectoarele de pe piață, iar domeniul IT nu face excepție. Întradevăr, cererea pentru specialiștii care termină o facultate cu profil tehnic este mare, dar și așteptările din partea angajatorilor sunt pe măsură. Într-un proces de angajare, mai ales pe pozițiile tehnice, experiența anterioară, studiile și cunoștințele sunt cele care primează. Astfel, pentru un proaspăt absolvent sau chiar un student, găsirea unui loc de muncă devine complicată, întrucât cunoștințele din facultate nu sunt suficiente pentru cerințele postului sau lipsa de experință devine un minus. În plus, chiar și obținerea unui interviu devine dificilă, din cauza concurenței.

Dacă totuși candidatul a obținut un interviu, vor urma mai multe etape de recrutare, fiecare fiind eliminatorie. Pentru pozițiile tehnice se preferă în general testarea candidaților folosind teste tehnice, fie pe hârtie, fie la calculator sau susținerea unor interviuri face-to-face, cu o persoană tehnică. În aceste tipuri de interviuri se urmăresc abilitățile candidatului de a răspunde la întrebări cât mai logic, de a rezolva probleme complexe de algoritmică și programare și de a cunoaște noțiunile teoretice din anumite limbaje de programare specifice postului. Evident, cu cât experiența candidatului este mai vastă, cu atât interviurile devin mai complicate și acoperă o gamă mai largă de subiecte.

Conform unor studii realizate de recrutori din domeniul IT, 50% dintre candidați nu trec de interviurile tehnice, iar motivele sunt numeroase:

1. Candidatul nu este pregătit din punct de vedere al cunoștiițelor.
2. Emoțiile sunt prea mari atunci când are loc un interviu cu o persoană experimentată.
3. Testele sau întrebările sunt prea grele pentru nivelul de pregătire al candidaților
4. Răspunsurile lor sunt ambiguue. Acest lucru indică nesiguranță, fapt care implică o pregătire slabă.

5. Răspunsul “nu știu” la întrebările puse. Acest lucru duce din start la pierderea oricărui avantaj, deoarece nimeni nu vrea să audă faptul că nu știi.
6. Timpul mult prea scurt până la susținerea interviului. Dacă între contactarea candidatului și interviul propriu zis sunt mai puțin de două zile, timpul de învățare va fi prea scurt, fapt care va duce la o pregătire foarte slabă.

Pentru un proaspăt absolvent, oricare dintre aceste motive poate să fie suficient pentru pierderea unui potențial job.

Astfel, tema aleasă în vederea pregătirii lucrării de licență își propune să ajute candidații să treacă cu brio de interviurile tehnice. Aplicația, denumită InterviewAssistant, este o aplicație mobilă, ce rulează pe dispozitivele cu sistem de operare Android. Scopul acesteia este de a oferi utilizatorilor o platformă educativă, fără a fi nevoie să caute în diferite surse de pe internet și să piardă timp verificând veridicitatea acestora.

Faptul că este o aplicație ce rulează pe dispozitivele mobile reprezintă un avantaj, deoarece utilizatorul are acces la ea în orice moment, chiar și fără o conexiune la internet.

Utilitatea ei provine din faptul că este personalizată în funcție de dorințele utilizatorului. De exemplu, să spunem că un candidat aplică pentru un internship, el fiind încă student, fără o experiență anterioară, iar postul îi solicită să știe ca limbaj de programare principal Java. El va completa toate aceste date într-un formular, iar apoi va parcurge întrebările tehnice ce îi vor fi generate pe baza mențiunilor completate. Astfel, dacă job-ul va fi un internhip, iar limbajul de programare va fi Java, utilizatorul va primi întrebări doar pentru nivelul de cunoștiințe selectat.

Scopul acestui filtru de generări este de a simplifica experiența utilizatorului, în sensul că la o simplă căutare pe Google de forma: întrebări interviuri tehnice Java, motorul de căutare îi va genera o multitudine de pagini web, cu o mulțime de întrebări tehnice, iar candidatul nu va ști care sunt cele mai potrivite pentru job-ul pentru care aplică. Personalizarea generării de întrebări va duce la minimizarea timpului alocat căutării, deoarece fiecare întrebare va avea și un răspuns unic aferent.

Totodată, după parcurgerea întrebărilor, utilizatorul poate opta să le păstreze ca favorite, pentru a le putea accesa oricând dorește pe cele preferate. Candidatul va mai avea opțiunea de a realiza quiz-uri, pentru a își testa cunoștiințele după parcurgerea întrebărilor. La fel, quiz-urile vor fi generate doar pentru limbajul de programare selectat, dar dacă utilizatorul dorește, poate exista opțiunea de a primi quiz-uri compuse din diferite limbaje. Fiecare întrebare va avea trei răspunsuri, dintre care doar unul va fi corect. Trecerea la următoarea întrebare va fi condiționtă

de răspunsul corect de la cea precedentă. La final, utilizatorul va primi punctajul obținut și va putea să își țină o evidență a evoluției în timp a punctajelor pe baza unor grafice.

Prin această aplicație se încearcă rezolvarea motivelor pentru care candidații nu reușesc să ajungă în următoarea etapă a procesului de recrutare. Ea este utilă mai ales atunci când nu există timp pentru a te pregăti și ai nevoie de o soluție rapidă și eficientă de învațat. În plus, ea este potrivită și celor aflați la început de drum, care vor să învețe bazele teoretice ale unui limbaj de programare. Faptul că telefonul mobil este dispozitivul cel mai utilizat în viața de zi cu zi, el permite acesul la date în orice moment, iar prin această aplicație se reduc nenumăratele căutări pe internet, care uneori reprezintă pierdere de timp. Se poate spune că toate informațiile de pe internet produc confuzie, deoarece niciodată nu este cert care este o sursă de încredere sau nu. Multitudinea de soluții contradictorii duce la alte căutări, până la găsirea rezultatului corect.

De aceea, este bine că această aplicație oferă răspunsurile corecte la întrebările generate, ajutând candidatul în obținerea job-ului visat sau fiind un pas premergător în procesul de învățare al unui nou limbaj de programare.

1.2 PREZENTAREA FUNCȚIILOR DE BAZĂ ALE UNEI APLICAȚII TIPICE

Orice aplicație mobilă, indiferent că rulează folosind un sistem de operare Android sau IOS, trebuie să respecte câteva funcții de bază, pentru a putea face interacționarea cu utilizatorul mai ușoară:

LOGIN

Accesul în aplicație trebuie să se facă printr-un sistem de tip login, unde utilizatorul își va introduce credențialele pentru a intra în aplicație. Acestea trebuie sa fie neapărat un username (de tip nume, adresa de email sau număr de telefon) și o parolă.

LOGOUT

Asigurarea delogării se face prin mesaje transmise utilizatorului, pentru ca acesta să fie sigur dacă vrea să părăsească aplicația sau nu.

PERSISTENȚA DATELOR

Aplicația trebuie să permită stocarea datelor la nivel de bază de date sau la nivel local.

CREARE CONT

În cazul în care utilizatorul nu are un cont pentru a se loga, aplicația va permite realizarea unui cont nou, unde se vor seta username-ul și parola pentru o viitoare logare.

MODIFICARE DATE CONT

Aplicația trebuie să permită modificarea credențialelor, dacă utilizatorul va dori la un moment dat acest lucru.

ACCESARE MENIU

Dupa logare, utilizatorul trebuie să aibă acces la un meniu contextual, pentru navigarea între activitățile aplicației.

PERMISIUNI

Setarea permisiunilor de care are nevoie aplicația: de exemplu, conexiunea la internet, accesul la sms, accesul la galeria de imagini, accesul la baza de date, accesul la camera foto.

ACTUALIZĂRI

Actualizarea aplicației trebuie să se producă real-time. Asta înseamnă că utilizatorul nu va trebui să acceseze serviciul Google Play și să facă actualizarea manual, de fiecare dată când se produc modificări de către dezvoltator.

SECURIZARE

Dacă se oferă acces la un browser web, aplicația trebuie să analizeze conținutul paginilor astfel încât să se asigure securitatea acesteia. În plus, trebuie să existe un mecanism de criptare a datelor, astfel încât informațiile despre utilizator să fie protejate.

COOKIE-URI

Stocarea cookie-urilor trebuie să fie permisă, pentru completarea automată a unor câmpuri.

COMPATIBILITATE

Aplicația trebuie să fie compatibilă cu orice versiune a sistemului de operare.

INTEROPERABILITATE

Dacă este cazul, aplicația trebuie să fie ușor de integrat cu anumite programe.

PERSONALIZARE

Aplicația trebuie să poată fi personalizată în funcție de cerințele utilizatorului.

1.3 PREZENTAREA UNOR SOLUȚII CONSACRATE PE PIAȚĂ

În acest subcapitol vor fi prezentate unele dintre aplicațiile deja existente pe piață, dezvoltate folosind platforma AndroidStudio și care oferă într-o oarecare măsură unele dintre funcționalitățile aplicației InterviewAssistant. Multe dintre aplicațiile disponibile pe GooglePlay sunt realizate fie de programatori juniori, care vor să capete experiență în dezvoltarea aplicațiilor mobile, fie de către firme specializate, care propun aplicații complexe, atât ca design cât și ca funcționalitate.

PROGRAMMING HUB

Programming Hub este o aplicație Android, dezvoltată de către Nexino Labs Pvt Ltd, ce propune învățarea celor mai cunoscute limbaje de programare: C, C++, JAVA, HTML, JAVASCRIPT, R, CSS, C#, SQL, PYTHON, LINUX, atât prin teorie cât și prin exemple de cod. Ca și funcționalități oferă un compiler pentru scrierea codului, cursuri teoretice, întrebări pentru interviurile tehnice, partajarea programelor scrise cu alți utilizatori ai aplicației, actualizări periodice, interfață disponibilă în 13 limbi, transmitere de feedback și peste 1800 de exemple de cod. Aplicația implementează și un modul de logare al utilizatorului, care presupune accesul la conturile deja existente pe telefon. Datele sunt stocate pe cardul SD, pentru ca aplicația să poată funcționa și în modul offline.

ITECHQUIZ

ITechQuiz este o aplicație Android ce oferă seturi de întrebări dispuse pe limbaje de programare. Oferă peste 600 de întrebări de interviuri pentru: Java, Java2EE, Servlets, Struts, Unix, SQL, Spring, Hibernate, .NET, C#, WCF, AJAX, XML, WebServices, precum și posibilitatea ca utilizatorul să adauge întrebări.

PROGRAMMING INTERVIEW QUESTIONS

Această aplicație dezvoltată de către Aurora Apps oferă întrebări pentru interviurile tehnice, dar în funcție de compania la care candidatul aplică. Astfel, utilizatorul poate să aleagă din peste 400 de companii (printre care Google, Amazon, Microsoft), peste 80 de topicuri despre limbaje de programare, precum și peste 70 de anunțuri de joburi la care poate să aplice. Aplicația oferă suport pentru concepte precum: algoritmi, structuri de date, sisteme de operare, baze de date, rețelistică și o multitudine de limbaje de programare.

2. PROIECTAREA APLICAȚIEI

Proiectarea se referă la transpunera vizuală a aplicației folosind un limbaj de modelare, denumit UML. Unified Modelling Language are scop modelarea proceselor business, analiza, design-ul și implementarea soluțiilor software. Modelarea UML a fost dezvoltată de către Object Management Group (OMG), iar prima variantă UML 1.0 a fost propusă în anul 1997. UML este în general folosit de către analiștii de business, arhitecții și dezvoltatatorii de produse software pentru a descrie și documenta fie procese business deja existente, fie unele noi. Modelarea poate să fie aplicată în diverse domenii, precum cel bancar, financiar, sănătate sau aerospațial și poate să fie implementată folosind diferite platforme, precum .NET. [1]

Modelarea UML presupune realizarea de diferite diagrame pentru o înțelegere mai ușoară a produsului. Aceste diagrame au un impact mai ridicat asupra înțelegerii, deoarece prin simplitatea lor se pot înțelege sisteme foarte complexe. Desenarea diagramelor UML se poate face folosind software specific, de exemplu programele Visio sau BPMN.io. Diagramele UML pot fi folosite atât de cei care dezvoltă produse software, cât și de utilizatorii obișnuiți sau de către oricine este interesat să afle cum funcționează sistemul. Sistemul se poate referi atât la produse sofware cât și la produse non-software. Trebuie înțeles faptul că modelarea UML nu este o metodă de dezvoltare, ci este în strânsă legătură cu sistemul pentru a îl face de succes. UML nu este un limbaj de programare, dar se pot folosi tool-uri pentru generare de cod în diferite limbaje.

Totodata, modelarea este strâns legată de conceptele programării orientate obiect și anume: abstractizarea, polimorfismul, moştenirea, encapsularea, clasele şi obiectele. Scopul analizei orientate obiect este de a analiza sisteme deja existente, folosind principiile enumerate mai sus. Prin aceasta se pot identifica obiectele sistemului, relaţiile dintre ele, dar şi realizarea design-ului folosind limbaje orientate obiect, precum Java sau C++.

Diagramele se împart în două mari categorii: structurale și comportamentale. Diagramele structurale definesc comportamentul static al sistemului, pe când cele comportamentale definesc comportamentul dinamic. Efectul vizual al diagramelor este cea mai importantă parte din întreg procesul, deoarece în acest fel se defineşte perspectiva asupra sistemului.[2]

[1] http://www.uml-diagrams.org/

[2] https://www.tutorialspoint.com/uml/uml_quick_guide.htm

În versiunea curentă UML 2.4.1 există 14 tipuri de diagrame dispuse astfel:

- Diagramele structurale din care fac parte: diagrama de clase, diagrama de componente, diagrama de implementare, diagrama de obiecte.
- Diagramele comportamentale din care fac parte: diagrama de pachete, diagrama profilelor, diagrama de structură, diagrama cazurilor de utilizare, diagrama de activități, diagrama de secvențe, diagrama de comunicare, diagrama de interacțiune, schema de timp, diagrama de stare.

În funcție de tipul de diagramă, fiecare are elemente specifice care ajută la alcătuirea ei. De exmeplu, diagrama de clase conține numele clasei, atributele, metodele. Diagrama cazurilor de utilizare conține diferiți actori care interacționează cu sistemul. Diagramele de activități și de stare au desenate noduri inițiale și finale, pentru a determina când este încheiat sau început procesul. Diagrama de componente conține produsele software și hardware care participă la dezvoltarea produsului. Diagrama de pachete este alcătuită în general din numele pachetelor care compun aplicația.

Oricare dintre aceste diagrame este completă doar dacă între componetele ei există și relații care să le lege. O relație între componente dă un anumit sens modelului UML.[3] Există patru tipuri diferite de relații în UML și anume:

- Relația de dependință, care descrie modul în care componentele unei diagrame depind unele de altele. Semnul acestei legături este o săgeată punctată, în care capătul din stânga reprezintă elementul dependent, iar cel din dreapta elementul independent.
- Relația de asociere, care descrie felul în care componentele sunt asociate. Legătura este reprezentată de o linie punctată cu săgeți în ambele capete.
- Relația de extensibilitate, folosită pentru a adăuga comportamente noi sistemului.
- Relația de generalizare, prin care se descrie conceptul de moștenire din programarea orientată obiect.

Prezentând cele de mai sus, se poate concluziona faptul că modelarea UML are scop simpla înţelegere a sistemelor complexe prin diferite diagrame, pentru diferiţi utilizatori. După multe versiuni ale limbajului, aceasta a devenit standardul OMG.

[3] https://www.tutorialspoint.com/uml/uml_basic_notations.htm

2.1 MODELUL DE DATE AL BAZEI DE DATE

Deoarece aplicația folosește o bază de date nerelațională, datele se află stocate în format Json, ca perechi cheie-valoare. Structura bazei de date conține 29 de noduri părinte, dintre care 27 reprezintă limbajele de programare denumite simbolic “CIntern”, “JavaIntern”, “HtmlMiddle”, “JavaScriptEntry” etc. Fiecare nod părinte conține noduri copil, identificate prin numere de la 1 la n. În nodurile copil de află stocate obiectele de tip cheie-valoare, fiind disponibile două chei: întrebare și răspuns. Fiecare cheie stochează întrebarea și răspunul aferent.


```
    },
"JavaEntry" : {
    "80" : {
      "intrebare" : "What is JDBC ?",
      "raspuns" : "JDBC is an abstraction layer that allows users to choose between databases."
    },
"JavaIntern" : {
    "97" : {
      "intrebare" : "What is JVM ? ",
      "raspuns" : "A Java virtual machine (JVM) is a process virtual machine that can execute Java bytecode. Each Java source file is compile
    },
"Likes" : {
    "CIntern" : [ {
      "intrebare" : " What is a class?",
      "raspuns" : "Class is a blue print which reflects the entities attributes and actions. Technically defining a class is designing an user
      "userId" : "IyPsNcnKJBTmoK56r9tVIKQzo1z2"
    }, {
      "intrebare" : "What is function overloading?",
      "raspuns" : "Defining several functions with the same name with unique list of parameters is called as function overloading.",
      "userId" : "IyPsNcnKJBTmoK56r9tVIKQzo1z2"
    }],
 "Users" : {
    "IyPsNcnKJBTmoK56r9tVIKQzo1z2" : {
      "email" : "test@gmail.com",
      "name" : "test"
    },
}
```


Figura 2.1.1- Exemplu de date din baza de date

Din figura 2.1.1, se pot observa nodurile părinte “JavaEntry”, “JavaIntern”, fiecare având copiii identificați prin numerele 80 și 97. În acest exemplu se află stocate și datele ce vor fi folosite în aplicație, date de forma întrebare-răspuns. Tipurile de date aferente cheilor sunt de tip String. Astfel, pe baza acestui exemplu este contruită toată baza de date.

Ultimele două noduri părinte sunt denumite “Likes” și “Users”. În nodul Likes se rețin toate întrebările la care un utilizator a dat like, iar în nodul Users se rețin toți utilizatorii aplicației si datele acestora: id-ul, email-ul și numele. Făcând o legătură cu bazele de date relaționale, se poate spune că între aceste două noduri există o legătură many-to-many, deoarece în nodul Likes se rețin id-urile utilizatorilor care au dat like la o anumita întrebare. Din figura 2.1.1 se observă că utilizatorul cu id-ul “IyPsnNcnKJBmoK56R9tVIKzo1z2” a dat like la două întrebări din limbajul de programare “CIntern”. Dacă un alt utilizator cu un alt id va da like la aceleași întrebări, se va crea o altă cheie “userId” care va stoca id-ul utilizatorului.

2.2 DIAGRAMA CAZURILOR DE UTILIZARE

Diagrama cazurilor de utilizare sintetizează aspectul dinamic al sistemului. Scopul acestei diagrame este de a aduna informații despre sistem și despre influențele externe și interne. Atunci când sistemul analizează datele, diagramele identifică actorii și modelează vizualizarea din afară. Actorii care sunt definiți sunt cei care interacționează cu sistemul. Acești actori pot fi: utilizatori, aplicații interne sau aplicații externe. Cand se realizează o astfel de diagramă, se ține cont de 3 aspecte: funcționalitățile să fie evidențiate, actorii să fie reprezentați și să existe relații definite între actori și funcționalități.

În continuare, este prezentată diagrama generală a cazurilor de utilizare pentru aplicația realizată:

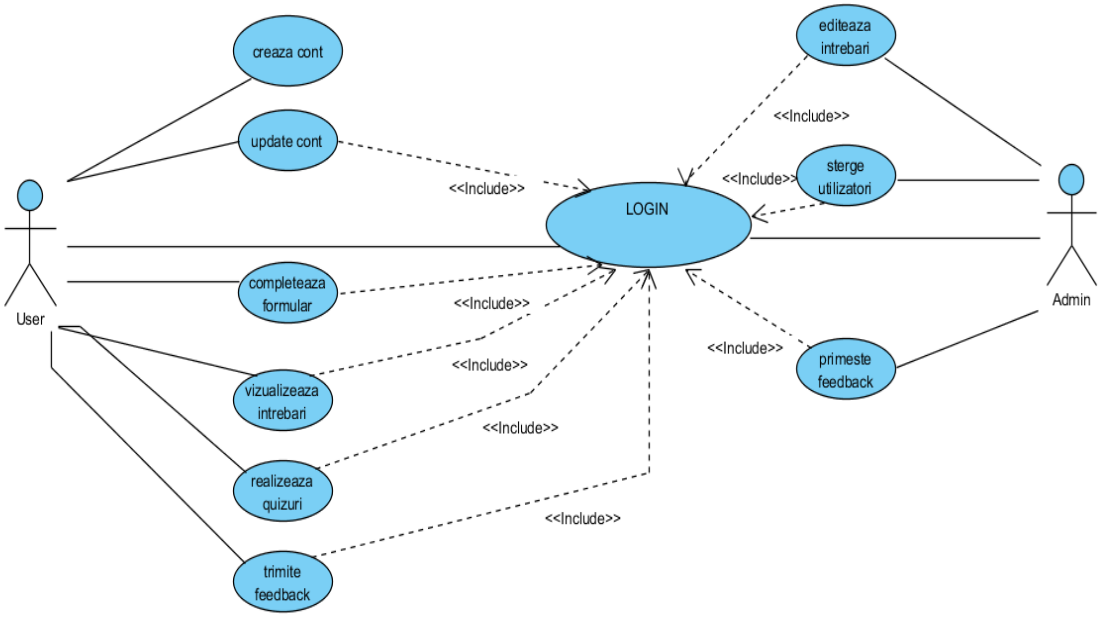

Figura 2.2.1- Diagrama generală a cazurilor de utilizare

Din figura 2.2.1, se poate observa că principalii actori ai sistemului sunt utilizatorul și administratorul aplicației, iar fiecare are rolurile lui bine definite. Utilizatorul poate să își creeze un cont, să modifice contul, să completeze formulare, să realizeze quiz-uri, să vizualizeze întrebări și să trimită feedback admin-ului. Aceste activități sunt strâns legate de activitatea de login, fără de care utilizatorul nu are acces în aplicație. Administratorul are drepturile lui speciale și anume: poate să editeze întrebările, să revoce drepturile utilizatorilor sau să primească feedback-ul. Toate acestea sunt posibile tot printr-o activitate de logare, admin-ul având acces la baza de date prin conectarea cu contul de Google.

2.3 DIAGRAMA DE ACTIVITATE

Diagramele de activități sunt folosite tot pentru a evidenția comportamentul dinamic al sistemului, dar, spre deosebire de restul diagramelor, scopul lor este de a realiza un flow, o legătură, între fiecare activitate. După identificarea corectă a activităților, ele trebuie corect desenate, iar acest lucru se poate realiza doar după o înțelegere a modului de comunicare între fiecare activitate. Astfel, înainte de a începe desenarea diagramelor, trebuie identificate următoarele elemente: asocieri între activități, condiții și constrângeri.

În figurile care urmează, sunt evidențiate aceste elemente și comunicarea între fiecare diagramă de activitate.

DIAGRAMA DE ACTIVITATE PENTRU CREARE CONT

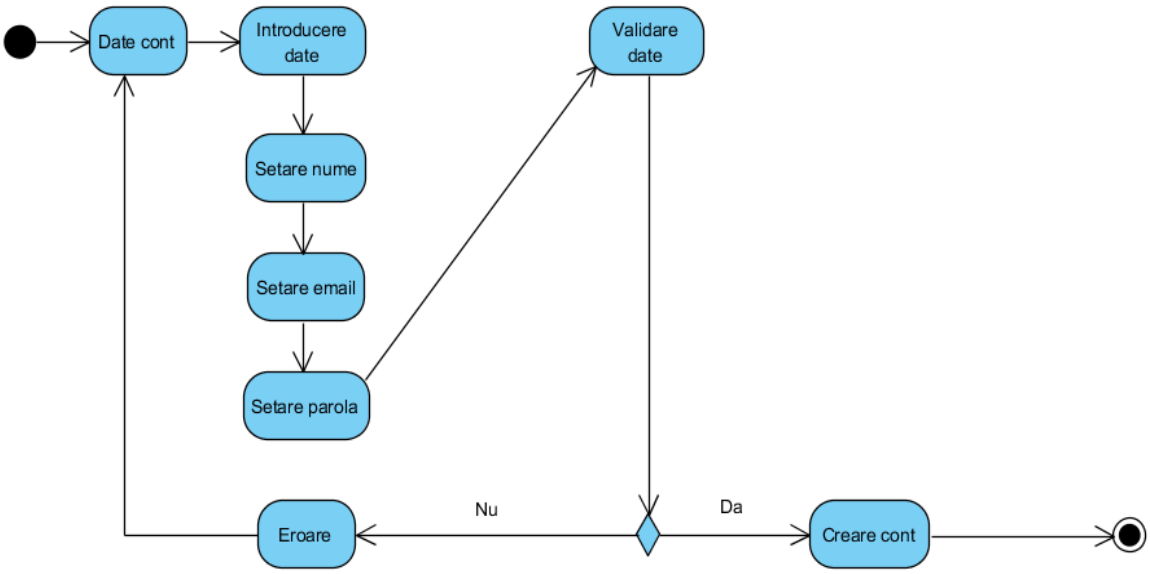

Figura 2.3.1- Diagrama de activitate pentru cazul creare cont

DIAGRAMA DE ACTIVITATE PENTRU TRIMITERE FEEDBACK

Figura 2.3.2- Diagrama de activitate pentru cazul trimitere feedback

DIAGRAMA DE ACTIVITATE PENTRU COMPLETARE FORMULAR

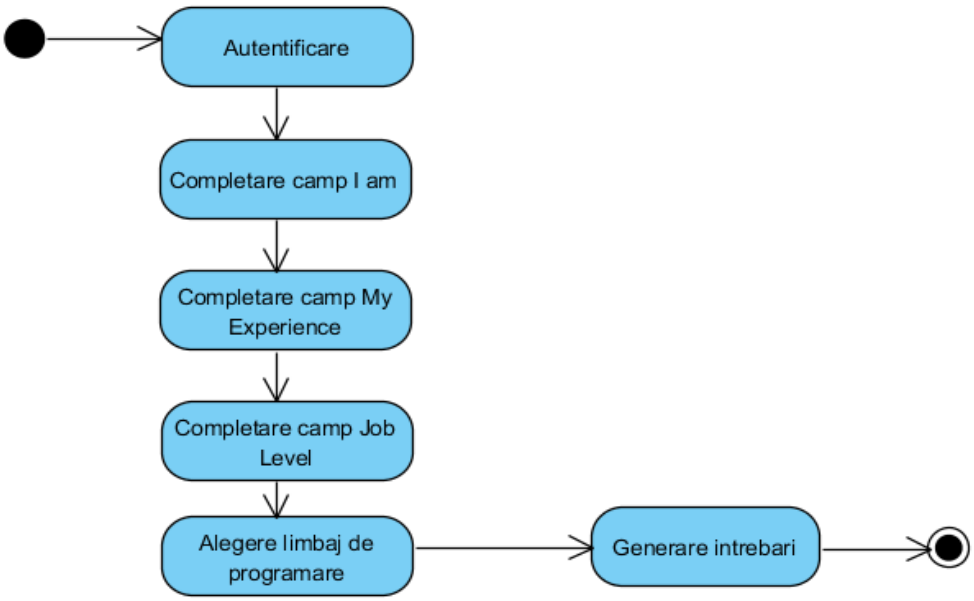

Figura 2.3.3- Diagrama de activitate pentru cazul completare formular

DIAGRAMA DE ACTIVITATE PENTRU VIZUALIZARE ÎNTREBĂRI

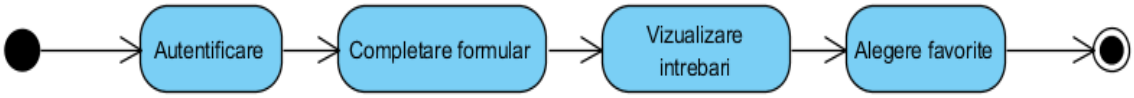

Figura 2.3.4- Diagrama de activitate pentru cazul vizualizare întrebări

DIAGRAMA DE ACTIVITATE PENTRU REALIZARE QUIZ

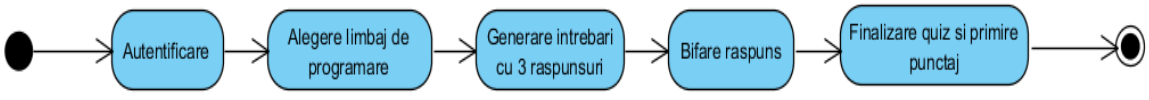

Figura 2.3.5- Diagrama de activitate pentru cazul realizare quiz

2.4 DIAGRAMA DE CLASE

Scopul diagramei de clase este de a evidenția clasele care intră în construirea aplicației, precum și a atributelor și metodelor specifice pentru fiecare clasă. Un rol important al acesteia este și faptul că analizează aplicația și descrie responsabilitățile sistemului. Practic, diagrama de clase reprezintă sistemul aplicației.

Desenarea ei constă în urmatoarele elemente:

- identificarea numelui clasei;
- identificarea corectă a atributelor și metodelor fiecarei clase
- pentru fiecare clasă se determină un număr minim de proprietăți, deoarece proprietățile inutile vor face diagrama complicată.

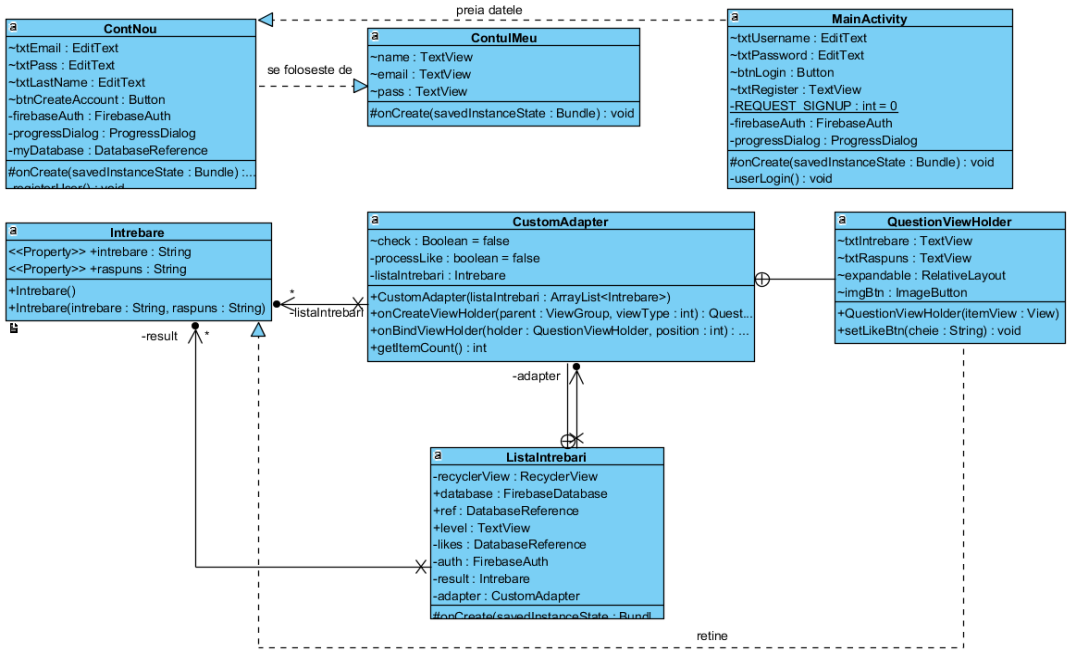

Figura 2.4.1- Diagrama de clase

În figura 2.4.1, se poate observa diagrama de clase, fiind reprezentate principalele activități cu care utilizatorul intră în contact: MainActivity, ContNou, ContulMeu, Întrebare, CustomAdapter, ListaIntrebări, QuestionViewHolder. Se poate observa că fiecare activitate are o metodă comună, și anume onCreateView(), metodă care se apelează la crearea fiecărei activități în parte. Activitățile ContulMeu și ContNou sunt legate între ele printr-o legătură 1-1, ceea ce arată că anumite cămpuri din activitatea ContNou vor fi vizibile în activitatea ContulMeu, dar la nivel unic de utilizator.

2.5 DIAGRAMA DE COMPONENTE

Diagrama de componente este folosită pentru a evidenția principalele relații între componentele sistemului și modul în care acestea interacționează. Diagramele de componente sunt desenate pentru a avea o privire de ansamblu asupra fiecărei componente a sistemului. Inițial, în UML 1.1, aceste componente erau reprezentate de: documente, tabele de date sau fișiere executabile. Începând cu UML 2.0, componentele diagramei au fost definite ca fiind: interfețetele, porturile, dispozitivele, dependințele sau serverele.

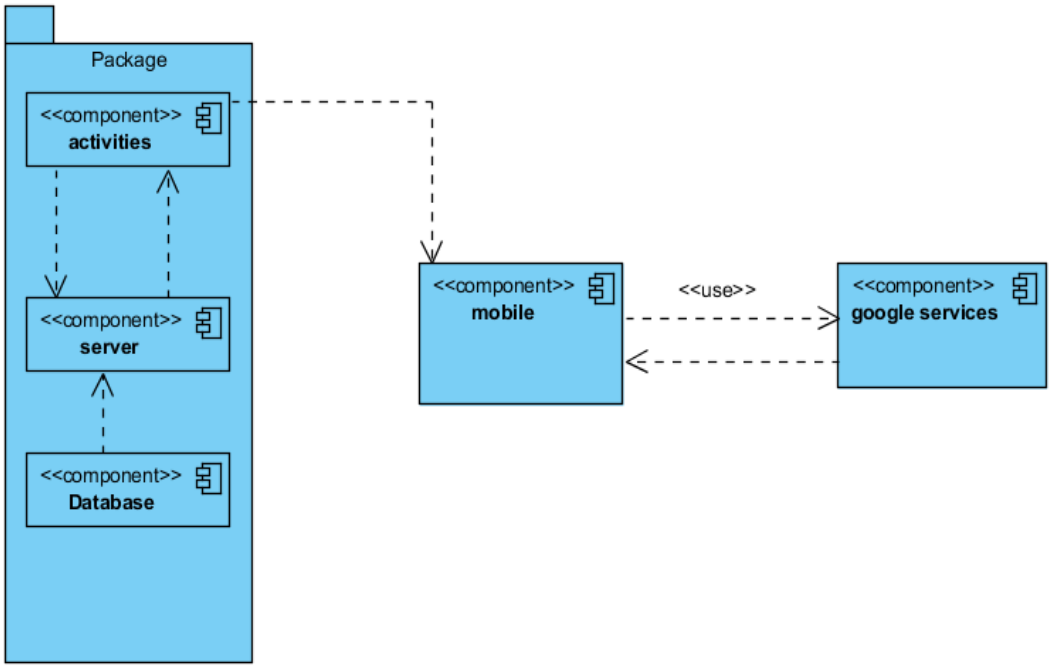

Figura 2.5.1- Diagrama de componente

În figura 2.5.1, se observă că principala componentă a aplicației o reprezintă telefonul mobil, dipozitivul pe care aplicația va rula. Cu acest dispozitiv interacționează restul de componente, precum:

- serviciile Google, care oferă acces la Google API sau Google Play.
- pachetul aplicației, .apk, care se dezvoltă sub forma de arhivă la crearea aplicației în AndroidStudio și în care se regăsesc activitățile, legătura cu baza de date și legătura cu serverul, clasele aplicației, fișierele ce conțin layout-ul activităților și fișierele de resurse în care se stochează imaginile.

2.6 SECURIZAREA APLICAȚIEI

Securizarea aplicației reprezintă procesul prin care dezvoltatorul, prin anumite funcționalități software sau hardware, protejează aplicația de eventualele amenințări externe. În ziua de azi, securizarea a devenit o problemă extrem de importantă atunci când are loc dezvoltarea unei aplicații, deoarece aceasta devine vulnerabilă, mai ales prin conexiunile WiFi sau prin descărcarea unor aplicații care pot prelua datele personale din telefon, folosind cod neautorizat.

Firebase asigură un set de funcționalități pentru a administra securitatea aplicației, denumite reguli, care determină cine are acces de scriere și citire la baza de date, cum sunt structurate datele sau ce indecși există. În Firebase, aceste reguli se numesc Firebase Realtime Database Rules.

Regulile se află stocate pe server și sunt accesate automat în orice moment. Solicitările de citire/scriere vor fi finalizate doar dacă regulile permit acest lucru. În mod implicit, aceste reguli sunt setate să permită citirea și scrierea completă în baza de date doar utilizatorilor care sunt autentificați. Acest lucru se întâmplă pentru a proteja baza de date de posibile abuzuri, până în momentul în care administratorul modifică regulile.

Regulilele sunt scrise în JavaScript și sunt de patru tipuri, după cum urmează:

- **.read:** determină când și dacă datele pot fi citite de utiizatori
- **.write:** determină când și dacă datele pot fi scrise de utilizatori
- **.validate:** determină modul în care va arăta o valoare
- **.indexOn:** determină cum sunt legate nodurile copil pentru a permite ordonarea și interogarea

Un prim pas în securizarea aplicației îl reprezintă identificarea utilizatorilor. Acest procedeu se numește *autentificare*. Serviciul de autentificare din Firebase poate fi utilizat folosind Firebase Authentification. Acesta este un serviciu back-end, care permite autentificarea prin: email și parolă, număr de telefon, cont Facebook, Gmail sau Google+, autentificare customizată și autentificare anonimă.

Al doilea pas îl reprezintă permiterea accesului pentru utilizatori doar la anumite date. Acest procedeu se numește *autorizare*.

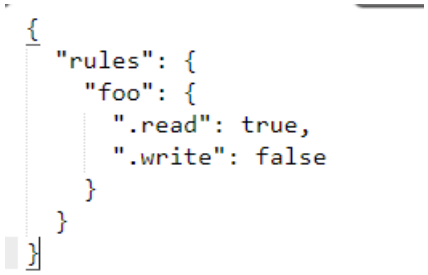

Figura 2.6.1- Reguli ".write" și ".read"

Regulile .read și .write sunt dispuse în cascadă, astfel încât ele asigură acces de citire pentru orice date aflate în directorul /foo, precum și a datelor stocate în alte directoare, dar aflate la calea /foo. Accesul de scriere este restricționat, deoarece regula .write are valoarea false.

Al treilea pas îl reprezintă *validarea datelor*. Această regulă restricționează utilizatorul să introducă doar câmpuri de un anumit tip și de o anumită lungime.

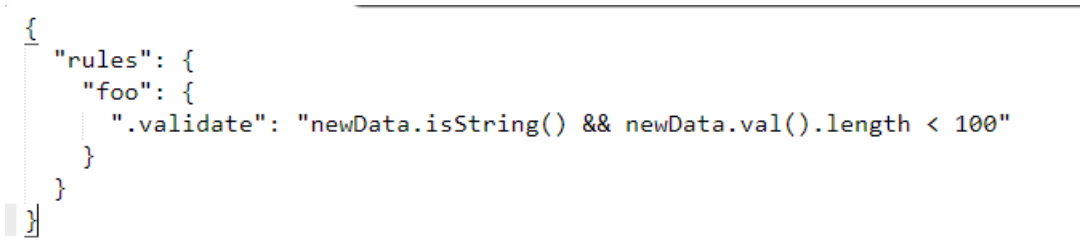

Figura 2.6.2- Regula ".validate"

Regula .validate nu diferă cu mult de .write și .read, doar că aceasta nu mai este dispusă în cascadă, astfel că toate regulile de validare trebuie să aibă valoarea true pentru a fi permisă regula de write.

Ultimul pas în securizarea aplicației este *indexarea*. Aceasta permite ordonarea și interogarea datelor. Acești indecși trebuie specificați pentru orice interogare și mai ales, înainte de rularea aplicației, pentru îmbunătățirea performanței query-urilor.

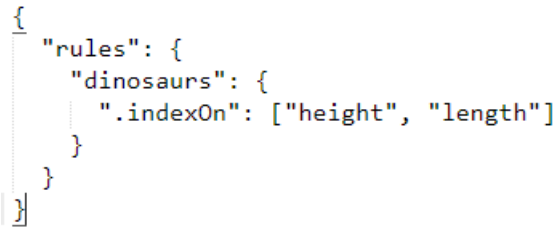

Figura 2.6.3- Regula ".indexOn"

Pe lângă regulile ce asigură securitatea, Firebase mai pune la dispoziție un mecanism de criptare a datelor, ce folosește protocolul SSL de 2048 biți. Acronimul provine de la Secure Socket Layer și reprezintă un standard de securitate care criptează datele transmise de la server la client. SSL permite transmiterea de informații, precum numărul de la card sau credențialele de la o logare, într-un mod securizat. Ca și protocol de securitate, el determină variabilele criptate atât pentru link cât și pentru datele transmise.

În mod normal, browserele comunică cu serverele web prin acest protocol. Asigurarea unei conexiuni sigure este posibilă prin folosirea unui certificat SSL. În general sunt disponibile două moduri de criptare a datelor folosind SSL: criptare simetrică și criptare asimetrică.

În Firebase, securizarea datelor se face prin criptare asimetrică. Acest mod de criptare, denumit și criptografie cu chei publice, se folosește de chei separate pentru criptare și decriptare. Astfel, orice poate folosi cheile publice pentru a cripta un mesaj, dar decriptarea este secretă. Cheile asimetrice sunt de obicei de 1024 sau 2048 biți, dar cele de 1024 biți nu mai sunt considerate sigure. De aceea, Firebase se folosește de cele de 2048 biți. Astfel, se spune că un calculator ar avea nevoie de aproximativ 14 miliarde de ani pentru a sparge un certificat de 2048 biți.[4]

Ca și mod de funcționare, comunicarea între server și browser se face astfel: serverul trimite o copie a cheii sale publice; browser-ul creează o cheie de sesiune simetrică pe care o criptează folosind cheia publică de la server. Apoi o trimite serverului. Serverul decriptează cheia simetrică criptată folosind cheia privată asimetrică. În acest moment, serverul și browserul criptează și decriptează datele transmise folosind cheia de sesiune. Aceasta permite transmiterea datelor într-un mod securizat, deoarece doar cele două părți cunosc cheia simetrică, care este valabilă doar pentru o sesiune. Pentru o următoare transmitere de date, folosind același server, o nouă cheie simetrică va fi creată.

Cel mai cunoscut algoritm ce utilizează criptarea asimetrică este RSA. Numele provine de la cei care au descris prima oară algoritmul, Ron Rivest, Adi Shamir și Leonard Adleman. RSA se bazează pe factorizarea numerelor întregi care sunt produse între două numere prime foarte mari. Decriptarea este aproape imposibilă, deoarece nu există un algoritm eficient pentru factorizarea întregului. Prin acest algoritm, capacitatea de criptare este strâns legată de lungimea cheii, astfel, o dublare a lungimii cheii va duce la creșterea rezistenței împotriva potențialelor atacuri, fără să afecteze performanța.

[4] https://www.digicert.com/ssl-cryptography.htm

3. IMPLEMENTAREA APLICAȚIEI

3.1 PREZENTAREA TEHNOLOGIILOR FOLOSITE

Pentru implementarea aplicației s-a folosit limbajul Java, precum și mediul de lucru AndroidStudio. La nivel de server s-a utilizat tehnologia Firebase, iar la nivel de stocare și utilizare de date s-a folosit o bază de date NoSQL.

Java este un limbaj de programare dezvoltat de către James Gosling pentru Sun Microsystems în 1995 ca și componentă a platformei Java 1.0. (cunoscută ca J2SE). Ca scop general, Java este un limbaj de programare bazat pe clase și orientat pe obiect și a fost dezvoltat pentru a avea cât mai puține dependințe posibile. Conceptul care a stat la baza dezvoltării acestuia a fost acela de a crea un limbaj care poate rula pe orice platformă care suportă Java, fără a fi nevoie de recompilarea codului. Astăzi, acest concept este cunoscut sub numele de „scrie o dată, rulează oriunde”. [5] Conform unor statistici realizate de Sun Microsystems, peste 3 miliarde de dispozitive rulează folosind Java, iar printre acestea se numără: aplicații desktop, web sau mobile, aplicații enterprise, sisteme embedded sau jocuri. De la lansare au fost implementate mai multe versiuni Java, versiunea curentă fiind JavaSE 8, lansată în 2014.

Aplicațiile Java sunt de obicei compilate la octeți, care pot rula pe orice mașină virtuală Java (JVM), indiferent de arhitectura calculatorului. Începând cu 2016, este cel mai utilizat limbaj de programare, folosit în special pentru aplicații client-server.[6] Din 2007, Java este o platformă open-source.

Limbajul a preluat în mare măsură sintaxa din C++, dar limbajele sunt diferite ca funcționalități. De exemplu, C++ suportă moștenirea multiplă prin clase, pe când Java nu permite acest lucru. Moștenirea se face folosind interfețe; C++ suportă supraîncărcarea operatorilor, iar Java nu suportă; C++ permite lucrul cu pointeri, pe când în Java acest lucru nu este posibil; C++ permite accesul la variabile atât prin valoare cât şi prin referinţă, iar în Java accesul se face doar prin valoare; C++ se bazează pe conceptul “scrie o dată, compilează oriunde”. Totuşi, există şi similtitudini între cele doua limbaje, cum ar fi faptul că ambele lucrează cu clase şi obiecte şi se folosesc de platforme independente.

[5] https://en.wikipedia.org/wiki/Java_(programming_language)

[6] Idem

În continuare vor fi prezentate 10 dintre cele mai importante funcționalități ale limbajului:

1. **Orientat obiect**- acest lucru înseamnă că se folosesc diferite obiecte la crearea softului care încorporează diferite comportamente. Printre principiile OOP utilizate în Java se numără: moștenirea, encapsularea, polimorfismul, abstractizarea, clasele și obiectele.
2. **Simplu de înțeles**- deoarece sintaxa a fost preluată din C++. Astfel, oricine a învățat C++ poate învăța foarte ușor și Java.
3. **Platformă independentă**- deoarece codul poate să fie rulat pe orice platformă, indiferent de sistemul de operare utilizat.
4. **Siguranță**- cu Java se pot dezvolta aplicații securizate, deoarece programele rulează folosind o mașină virtuală.
5. **Robust**- deoarece limbajul folosește o administrare a memoriei foarte puternică.
6. **Portabilitate**- deoarece poate rula pe orice sistem de operare.
7. **Performanță ridicată**
8. **Distribuire-** se pot dezvolta aplicații distribuite
9. **Multi-threading-** limbajul permite rularea mai multor task-uri în același timp folosind firele de execuție. În plus, multithreading-ul nu ocupă memorie pentru fiecare fir de execuție în parte.
10. **Dinamic-** Java este considerat un limbaj mai dinamic decât C++, deoarece este proiectat să se adapteze la un mediu în evoluție.[7]

Java nu este numai un limbaj de programare, cât și o platformă. Ca definiție generală, o platformă este acea componentă software sau hardware cu ajutorul căreia se rulează programele. Platforma Java diferă de majoritatea platformelor cunoscute, de exemplu Linux, Mac, Windows, deoarece este bazată pe un software care rulează pe lângă alte platforme hardware. Platforma Java utilizează două componente: JVM (java virtual machine) și API (java application programming interface). API-ul este o colecție de componente software care oferă multe funcționalități utile, grupate în librării. Aceste librării poartă denumirea de pachete. Deaorece este un mediu independent de platformă, platforma Java poate să fie uneori mai lentă decât codul nativ. Cu toate acestea, progresele înregistrare la nivel de compilator și mașină virtuală aduc performanțe apropiate de cele ale codului nativ, fără a periclita transferabilitatea.[8]

[7] https://www.tutorialspoint.com/java/java_overview.htm

[8] https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Android este un sistem de operare dezvoltat de către Google și bazat pe un sistem Linux cu mai mulți utilizatori, fiecare aplicație fiind reprezentată de un utilizator diferit. În mod implicit, sistemul asignează fiecărei aplicații un id unic. Acest id este folosit doar de sistemul de operare, fără ca aplicația să îl cunoască. Astfel, sistemul stabilește permisiuni pentru toate fișierele aplicației, dar numai id-ul aplicației poate să le acceseze.[9]

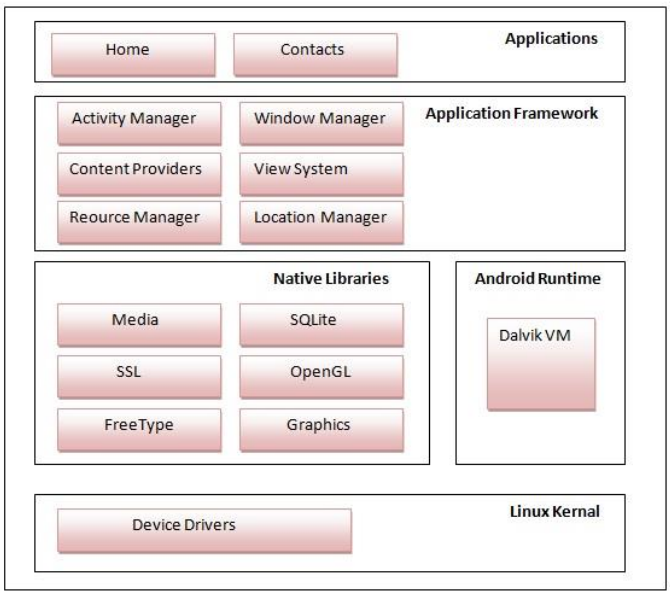

Figura 3.1.1- Arhitectura sistemului de operare Android

Arhitectura sistemului Android este împărțită în 4 nivele și în 5 secțiuni esențiale după cum urmează: linux kernel, librăriile, android runtime, application framework și aplicațiile. Prima secțiune, nucleul linux, oferă un nivel de abstractizare între hardware-ul dispozitivului și driverele acestuia, precum camera foto, display-ul, driver-ul bluetooth, wifi sau audio. Librăriile se află deasupra nivelului linux kernel și sunt compuse din librăriile ce stochează datele (Sqlite), librăriile SSL necesare pentru securitate, librăriile necesare pentru redarea de conținut audio și media. A treia secțiune reprezentată de android runtime conține librării ce permit dezvoltatorilor să scrie aplicații Android folosind cod Java. De asemenea, la acest nivel se află și mașina virtuală Dalvik, care permite fiecărei aplicații să ruleze propriul proces, fiind izolată de restul aplicațiilor. A patra secțiune, application framework, furnizează servicii către aplicație sub formă de clase Java. Principalele servicii disponibile sunt: Activity Manager, Content Providers, Resource Manager, Notification Manager, View System. În ultima secțiune se va instala aplicația realizată.

[9] https://developer.android.com/guide/components/fundamentals.html

Cele mai importante componente ale unei aplicații Android sunt în număr de cinci și anume: activități, servicii, furnizori de conținut, consumatori de intenții și intent-uri. Acestea nu trebuie să lipsească atunci când se dezvoltă aplicația, deoarece prin ele sistemul sau utilizatorul are acces la aplicație.

Activitățile reprezintă interfața cu utilizatorul. O aplicație poate să aibă mai multe activități, gestionate de sistem prin serviciul Activity Manager. Ciclul de viață al unei activități este independent de ciclul de viață al aplicației și este reprezentat de apelul metodelor: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy(), onRestart(). O activitate este instanțiată folosind o subclasă a clasei Activity.

Serviciile sunt componente care rulează în fundal, fără a interacționa cu utilizatorul și fără a pune la dispoziție o interfață. Un exemplu de serviciu ar putea fi rularea muzicii în fundal sau transmiterea de date fără a bloca interacțiunea utilizatorului cu activitatea. Un serviciu este creat ca instanță a clasei Service.

Furnizorii de conținut sunt componente ce permit sistemului să partajaze și să stocheze, date în sistem. Printr-un furnizor de conținut se pot modifica date, dacă aplicația permite acest lucru. Aceștia se implementează ca instanță a clasei ContentProvider.

Consumatorii de intenții permit sistemului să transmită evenimente în afara fluxului cunoscut de utilizator. De exemplu, notificările referitoare la starea bateriei sau la downloadarea unui fișier sunt transmise prin aceste componente, instanțiate folosind clasa BroadcastReceiver.

Un intent este o descriere a unei operații ce urmează să fie executată. Trecerea dintr-o activitate în alta sau transmiterea de date între activități se realizează prin intent. Acesta este instanțiat ca obiect al clasei Intent.

O altă componentă importantă a unei aplicații Android este fișierul Manifest.xml. Acesta este creat o dată cu aplicația și se află poziționat ca fișier rădăcină în structura proiectului. Rolul lui este de a specifica componetele de care are nevoia aplicația pentru a rula funcțional. De exemplu, în el se declară permisiunuile necesare aplicației, precum accesul la internet sau la camera foto dar și nivelul minim de API cerut de aplicație.

Resursele unei aplicații sunt componente separate de cod, aflate în directorul /res. Resursele pot fi: imagini, fișiere video sau audio, meniuri, culori, layout-uri, fișiere xml. Pentru stocarea diferitelor resurse, se folosesc foldere speciale precum: res/color, res/menu, res/xml, res/anim, res/drawable, res/layout, res/values, res/raw, res/animator. La crearea unei noi resurse, sistemul alocă fiecăreia un id unic, prin care poate fi identificată. Prin folosirea resurselor, actualizarea unor caracteristici ale aplicației se face mai ușor, fără a modifica codul.

Bazele de date NoSQL descriu un mecanism de stocare și preluare a datelor care sunt modelate prin alte mijloace decât cele bazate pe tabele, folosite de bazele de date relaționale. În ultimii ani, o dată cu evoluția internetului și a aplicațiilor web, bazele de date au crescut în dimensiuni, ajungând să stocheze milioane de teraBytes. Aplicații precum Facebook, Amazon sau Whatsapp necesită un volum mare de date, pentru care este nevoie de o bază de date puternică, mult mai rapidă decât bazele de date tradiționale.

În general, datele din bazele de date NoSQL sunt structurate ca perechi cheie-valoare. De obicei, datele pot să fie dispuse în fișiere xml sau Json. Pentru aplicația realizată s-a folosit stocarea în format Json. Datorită acestor formate, se reduce codul necesar convertirii datelor, ceea ce determină un cost mai scăzut al mentenanței.

Bazele de date non-relaționale aduc beneficii prin faptul că nu există o structură strictă a datelor. Ele au o tendință de dezvoltare dinamică și de aceea pot să suporte date semi-structurate, structurate și nestructurate. Spre deosebire de bazele de date relaționale, unde scalarea este verticală și se adaugă capacitate extinsă serverelor, bazele de date non-relaționare au o scalare orizontală. Acest lucru înseamnă că se pot adăuga mai multe servere prin procese de replicare. Totodată, performanța este îmbunătățită în comparație cu bazele de date relaționale.

După cum am prezentat mai sus, datele din baza de date sunt stocate în format Json. Acronimul provine de la JavaScript Oriented Notation și reprezintă un mod organizat de stocare a datelor după un anumit format. Datorită acestui lucru, este foarte ușor de citit pentru oameni și foarte ușor de parsat și generat de către calculatoare. Formatul Json este derivat din limbajul JavaScript, dar este independent de acesta. În scrierea lui se folosesc convenții care sunt familiare celor ce programează în limbajele C, C++, Java, Python, Pearl. De obicei acest format este folosit pentru transmiterea de date între server și aplicație, fiind o alternativă a fișierelor xml. În Json, un obiect este un set neordonat de perechi cheie-valoare, care este cuprins între acoladele { }. Fiecare cheie este urmată de ":", iar valorile sunt despărțite prin virgulă.

În figura 3.1.2 se poate observa un exemplu de obiect Json, în care cheile sunt reprezentate de „age", „name" și „gender", iar valorile de „24", „john" și „male".

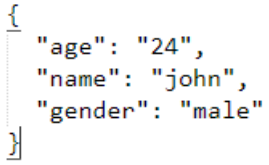

Figura 3.1.2- Exemplu de date Json

Firebase este o platformă pentru dezvoltarea aplicațiilor web sau mobile, cât și un serviciu back-end, oferind în principal o bază de date real time, un serviciu de găzduire și un serviciu de autentificare al utilizatorilor.

De-a lungul timpului, Firebase și-a extins linia de produse, ajungând astăzi să ofere următoarele servicii:

Realtime Database este un serviciu ce stochează și sincronizează datele între utilizatori și dispozitive în timp real, folosind baze de date NoSQL stocate în fișiere Json pe cloud. Avantajul acestui serviciu este că aplicația se conectează la Firebase prin websocket-uri și nu prin http, websocket-urile fiind mult mai rapide. Deoarece se efectuează un singur apel către socket, datele sunt sincronizate automat prin acel websocket. Firebase trimite datele pe măsură ce sunt actualizate. Astfel, când un client salvează o modificare, toți clienții conectați primesc actualizarea imediat.

Firebase Auth este un serviciu de autentificare al utilizatorilor prin email și parolă. De asemenea, suportă conectarea cu aplicații precum Google, Facebook, Twitter sau GitHub.

Firebase Cloud Storage permite stocarea de imagini, fișiere audio/video în fișiere binare, în Google Cloud.

Firebase Crash Reporting determină problemele aplicației prin realizarea de rapoarte ale bug-urilor, grupate în funcție de frecvență și de severitate.

Firebase Cloud Functions permite extinderea aplicației prin cod back-end personalizat, fără ca dezvoltatorii să gestioneze propriile servere.

Firebase Hosting este un serviciu web static de găzduire a fișierelor html, css, javascript, ce nu pot fi modificate dinamic.

Firebase Perfomance Monitoring este un serviciu ce verifică performanța aplicațiilor pe dispozitivele utilizatorilor.

Firebase Analytics oferă servicii de monitorizare a activității utilizatorilor.

Firebase Cloud Messiging trimite mesaje și notificări gratis utilizatorilor de pe platformele Android, Ios sau web.

Firebase Invites oferă utilizatorilor posibilitatea de a transmite prin email sau sms detalii despre aplicație cu alți utilizatori.

Firebase App Indexing este un serviciu ce permite aplicațiilor să fie vizibile în google search prin folosirea de indecși.

Firebase Remote Config este un serviciu ce permite modificarea design-ului aplicației fără ca utilizatorii să fie nevoiți să realizeze actualizări manual.

Cu toate că foarte mulți dezvoltatori consideră Firebase ca fiind o soluție foarte bună atunci când se realizează aplicații web sau mobile, aceasta vine cu avantajele și dezavantajele ei. Un prim avantaj este faptul că baza de date este real-time. Astfel, indiferent de ce modificare este făcută, actualizarea aplicației se produce automat.

Structurarea datelor în format Json duce la eliminarea conceptelor din bazele de date relaționale, precum tabele sau chei străine. Chiar dacă acest lucru ajută la standardizarea conținutului, există totuși probleme legate de interogări. Dacă într-o bază de date relațională datele sunt dispuse în tabele și sunt ușor de indexat și căutat folosind cheile străine, în Firebase pot exista milioane de înregistrări stocate fără o logică anume. Căutarea se relizează mai greu deoarece nu există o metodă implementată de Google care să suporte căutarea atunci când obiectele au legătură între ele.

Un alt avantaj îl reprezintă stocarea datelor în cloud. Accesul la date este asigurat permanent, cu mențiunea să existe o conexiune la internet. Autentificarea prin diferite metode este un plus, deoarece îi oferă utilizatorului opțiunea de a se loga folosind o varietate de conturi. În plus, folosind Firebase, dezvoltatorul nu trebuie să își mai implementeze propriul server, deoarece se folosește de serviciul cloud functions, care ține loc de server. Încă un avantaj este reprezentat de securizarea aplicației, deoarece Firebase pune la dispoziție o multitudine de metode și reguli prin care dezvoltatorul asigură securitatea maximă. Pe langă aceste reguli, se folosește și protocolul SSL pentru criptarea datelor. În plus, aplicațiile pot fi folosite offline, deoarece datele sunt sincronizate imediat atunci când se realizează conectivitatea.

Un ultim avantaj este acela că se realizează backup-uri permanente în locații securizate, pierderea datelor fiind astfel minimizată.

Totuși, cum am prezentat mai sus, Firebase are limitări în ceea ce privește interogarea datelor din cauza structurii acestora. De asemenea, agregarea datelor nu este posibilă și nici listarea utilizatorilor sau a fișierelor stocate. În plus, dacă se folosește un volum mare de date structurarea acestora este dificilă, mai ales dacă dezvoltatorul este obișnuit cu bazele de date relaționale. Un alt dezavantaj este faptul că platforma nu este folosită îndeajuns pentru aplicațiile enterprise. Totodată, utilizarea Firebase este gratis doar dacă sunt stocați mai puțin de 50 de utilizatori și mai puțin de 100mb de date. Migrarea datelor se realizează mai greu din cauza formatului Json.

Cu toate acestea, Firebase poate fi folosit ca și serviciu back-end pentru aplicațiile noi, dacă acestea nu folosesc un volum mare de date sau dacă nu utilizează servicii terțe. Aplicațiile deja existente pot fi extinse prin folosirea unor funcționalități noi. În plus, Firebase poate funcționa ca intermediar între client și server, folosind serviciile REST.

3.2 IMPLEMENTAREA APLICAȚIEI FOLOSIND TEHNOLOGIILE UTILIZATE

În acest capitol vor fi prezentate funcționalitățile care au stat la baza construirii aplicației. Deoarece este o aplicație Android, în dezvoltarea ei s-a utilizat mediul de lucru AndroidStudio, care permite crearea activităților ce interacționează cu utilizatorul.

La crearea unui nou proiect se stabilesc tipurile de activități ce vor fi folosite. În acest caz s-au folosit cinci activități de tip EmptyActivity: MainActivity, ContNou, ForgotPassword, ListaIntrebari, Favorite și o activitate de tip MenuActivity, denumită Meniu. Cu ajutorul activității de tip meniu se navighează în aplicație, folosind patru opțiuni: PageForm, PageQuiz, PageFeedback și PageMyAccount. Atunci când se crează o activitate nouă se crează automat un layout, cu ajutorul căruia dezvoltatorul va desena și gestiona elementele din interfața respectivă. Layout-urile se regăsesc în folderul /res/layout și au denumirea de forma "activity_cont_nou.xml".

La nivel de proiect, în fișierul Manifest.xml se setează permisiunile aplicaţiei, folosind tag-ul <*uses-permission android:name=" "*>. După cum se poate observa în figura 3.2.1, sunt setate accesul la internet, la contactele din telefon, precum și accesul de scriere folosind o sursă externă, de exemplu baza de date.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Figura 3.2.1- Setarea permisiunilor în Manifest.xml

După cum este prezentat în Anexa 1, în acest fișier se mai setează detalii despre aplicație, precum și informații despre fiecare activitate în parte. La nivel de aplicație, folosind tag-ul <*application*>, se pot seta numele aplicației și iconița ce va fi vizibilă când aplicația va fi descărcată pe telefon. Tag-ul <*application*> conține toate activitățile aplicației, evidențiate prin tag-ul <*activity*>. La nivel de activitate se pot seta numele activității, label-ul, tema, dar și opțiuni referitoare la tastatură. Prima activitate cu care interacționează utilizatorul atunci când deschide prima oară aplicația este setată folosind tag-ul <*intent-filter*>. În acest caz prima activitate va fi MainActivity, specificată folosind tag-ul <*action android:name=" "*>. Toate cele prezentate se află în interiorul tag-ului <*manifest*>, ce conține și numele pachetului aplicației.

Interfața este construită folosind o ierarhie de layout-uri și widget-uri. Layout-urile sunt containere invizibile care controlează felul în care sunt poziționate elementele pe ecran. Există două tipuri de layout-uri: RelativeLayout, în care componentele sunt așezate unul sub altul și LinearLayout, în care componentele sunt așezate unul în continuarea celuilalt. Widget-urile sunt componente de interfață și pot fi: butoane, edit text-uri, check-box-uri etc. Toate aceste elemente sunt setate de dezvoltator prin intermediul fișierelor xml.

Layout-urile activităților se regăsesc în folderul /res/layout. Indiferent de tipul de layout ales, el trebuie să conțină două specificații importante: înălțimea și lățimea. Acestea sunt setate folosind proprietățile *layout_width* și *layout_height* din figura 3.2.2. Valoarea "*match_parent*" va forța layout-ul să ocupe tot spațiul disponibil, iar valoarea "*wrap_content*" va forța layout-ul să fie destul de mare încât sa cuprindă toate elementele.

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

Figura 3.2.2- Setarea permisiunilor în Manifest.xml

În anexa 2 se poate observa codul xml pentru interfața activității MainActivity. Acesta conţine un layout de tip RelativeLayout. Fundalul este setat folosind proprietatea *"android:background"*. Widget-urile folosite sunt două edit text-uri, două text view-uri și un buton. Fiecare widget are asignat un id unic, folosind proprietatea "android:id". În figura 3.2.3 se poate observa că în clasa MainActivity elementele sunt identificate folosind metoda *findViewById()*, care primește ca parametru id-ul fiecărei componente.

```
btnLogin=(Button) findViewById(R.id.btn_login);
progressDialog = new ProgressDialog(this);
txtUsername = (EditText) findViewById(R.id.editTextUsername);
txtPassword = (EditText) findViewById(R.id.editTextPassword);
txtForgotPass=(TextView) findViewById(R.id.textView_ForgotPass);
```

Figura 3.2.3- Exemplu de identificare a componentelor în cod

În anexa 3 sunt evidențiate componentele interfeței ConNou și anume: trei controale de tip edit text și un control de tip buton. Accesarea acestei interfețe se face din MainActivity, printr-un eveniment de click pe text view-ul "Not an account yet?". Evenimentul se realizează prin apelarea metodei setOnClickListener, exemplificată în Anexa 6, iar trecerea dintr-o activitate în alta se face folosind o instanță a clasei Intent, care primește ca parametri contextul actual și contextul în care se dorește să se ajungă.

Anexele 4 și 5 reprezintă codul xml pentru interfața ForgotPassword, compusă din trei controale statice de tip text view, un edit text și un buton. Rolul interfeței este de a permite resetarea parolei utilizatorului, prin completarea edit text-ului și apoi printr-un eveniment de click pe buton.

Codul xml pentru prima opțiune din meniu, PageForm, conține trei controale de tip edit text, personalizate folosind tag-ul *<android.support.design.widget.TextInputLayout/>*, după cum este prezentat în anexele 6-7. Widget-ul de tip buton permite trecerea în activitatea ListăIntrebări. Edit text-urile sunt completate de utilizator fără scrierea de la tastatură, ascunderea acesteia fiind posibilă folosind metoda *„setShowSoftInputonFocus(false)”*. Datele sunt selectate din ferestre speciale numite AlertDialog, ce se deschid atunci când se selectează un edit text.

Interfața pentru opțiunea de trimitere feedback conține un control de tip edit text în care utilizatorul va completa mesajul pe care îl va trimite și un buton pentru trimiterea acestuia prin YahooMail sau Gmail. Modificarea datelor utilizatorului este disponibilă din interfața PageMyAccount, al cărei cod xml se află prezentat în anexa 8. Ea conține două edit text-uri, unul pentru schimbarea adresei de email și unul pentru schimbarea parolei și două butoane: de delogare și de aplicare a schimbărilor. Ambele evenimente pe butoane se realizează prin evenimentul setOnClickListener() și prin suprascrierea metodei OnClick().

Aceste interfețe folosite în meniu sunt implementate folosind conceptul de fragmente. Un fragment reprezintă un comportament într-o activitate. Astfel, se pot combina mai multe fragmente pentru a construi o interfață multiplă și pentru a reutiliza codul în alte activități. Un fragment este instanțiat folosind clasa Fragment. Adăugarea unui layout pe un fragment este posibilă folosind metoda *onCreateView()* și încărcând layout-ul din xml prin metoda *inflate*. Această metodă primește ca prim parametru numele layout-ului, după cum se observă în figura 3.2.4.

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Figura 3.2.4- Încărcarea unui layout într-un fragment

Întrucât baza de date se află stocată în cloud, prin intermediul serviciului Firebase, conectarea la aceasta se face prin adăugarea unor dependințe în fișierele build.gradle ale

aplicației și prin importarea fișierului google-services.json, care este automat generat de către Firebase.

```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.1.0'
    }
}
```

Figura 3.2.5- Adăugarea fișierelui google-services.json

Folosirea aplicației este condiționată de crearea unui cont de către utilizator. Crearea acestui cont presupune introducerea datelor personale (nume, adresă de email și parolă) în activitatea ContNou, date ce vor fi apoi salvate în baza de date. Firebase permite înregistrarea unui utilizator nou prin folosirea librăriei 'com.google.firebase:firebase-auth'.

Referința către baza de date a aplicației se realizează prin crearea unui obiect de tip DatabaseReference care accesează instanța curentă a bazei de date. Datele utilizatorului vor fi salvate în nodul "Users", accesat ca nod copil al bazei de date curente. Reținerea utilizatorului nou înregistrat se face prin crearea unui obiect de tip FirebaseAuth. În acest obiect se reține utilizatorul curent, prin instanța acestuia. În figura 3.2.6 se poate observa cum sunt instanțiate cele două obiecte, myDatabase și firebaseAuth.

```
myDatabase= FirebaseDatabase.getInstance().getReference().child("Users");
firebaseAuth = FirebaseAuth.getInstance();
```

Figura 3.2.6- Crearea obiectelor de tip DatabaseReference și FirebaseAuth

Metoda ce permite crearea unui utilizator nou este pusă la dispoziție de librăria firebase-auth și se numește *"createUserWithEmailAndPassword(email password)"*. Metoda este apelată atunci când se realizează evenimentul de click pe butonul "CREATE ACCOUNT". Dacă câmpurile nu sunt completate în totalitate sau sunt completate greșit, se vor genera erori la nivel de câmp, lucru evidențiat în anexa 15. Această metodă accesează instanța curentă de utilizator și primește ca parametri datele introduse de utilizator în activitate și anume email-ul și parola. Dacă task-ul este realizat cu succes, atunci un nou utilizator este creat, datele acestuia sunt salvate în nodul Users, iar logarea în aplicație se va face automat. În caz contrar, crearea contului poate eșua dacă există deja un utilizator cu același email sau dacă parola este invalidă. Figura 3.2.7 prezintă apelul metodei, precum și ce se returnează în cazul în care contul a fost creat sau nu a fost creat.

```
firebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(this, (task) → {
                if(task.isSuccessful()){
                    String userId=firebaseAuth.getCurrentUser().getUid();
                    DatabaseReference currentUser=myDatabase.child(userId);
                    currentUser.child("name").setValue(lastName);
                    currentUser.child("email").setValue(email);

                    Intent intent=new Intent(getApplicationContext(),MainActivity.class);
                    startActivity(intent);
                    Toast.makeText(ContNou.this,"Successfully registered",Toast.LENGTH_LONG).show();

                }else{
                    Toast.makeText(ContNou.this,"Registration Error",Toast.LENGTH_LONG).show();
                }
                progressDialog.dismiss();
        });
```

Figura 3.2.7- Crearea unui utilizator nou

În cazul în care utilizatorul are deja un cont creat, se va trece la activitatea de logare. Logarea se realizează din MainActivity și presupune introducerea credențialelor email și parolă corecte. Dacă datele nu sunt introduse corect se va genera un mesaj care va atenționa utilizatorul că una dintre valori este greșită.

```
//checking if email and passwords are empty
if(TextUtils.isEmpty(email) || TextUtils.isEmpty(password)){
    Toast.makeText(this,"Email or password incorect",Toast.LENGTH_LONG).show();
    return;
}
```

Figura 3.2.8- Verificare introducere date la logare

Implementarea evenimentului de logare folosind Firebase se realizează prin metoda *"signInWithEmailAndPassword(email, password)"*. Această metodă accesează utilizatorul curent folosind variabila firebaseAuth, care determină instanța unică de utilizator. Dacă credențialele sunt recunoscute cu cele din baza de date, iar task-ul este realizat cu succes, utilizatorul va fi redirecționat către activitatea Meniu folosind clasa Intent, după cum se observă în figura 3.2.9. În caz contrar, utilizatorul va primi un mesaj de eroare, iar autentificare nu se va realiza.

```
//logging in the user
firebaseAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(this, (task) → {
                progressDialog.dismiss();
                if(task.isSuccessful()){
                    finish();
                    startActivity(new Intent(getApplicationContext(), Meniu.class));
                }
                else {
                    Toast.makeText(MainActivity.this,"Email or password incorect",Toast.LENGTH_LONG).show();

                }
        });
```

Figura 3.2.9- Autentificare utilizator

Din activitatea MainActivity, utilizatorul își poate reseta parola, prin folosirea controlului de tip text view, “Forgot Password?”. La evenimentul de click pe acest control, utilizatorul este redirecționat către activitatea care îi va permite resetarea parolei. Implementarea evenimentului de click poate fi observat în figura 3.2.10.

```
txtForgotPass.setOnClickListener((v) → {
        Intent it=new Intent(getApplicationContext(),ForgotPassword.class);
        it.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(it);
});
```

Figura 3.2.10- Implementare eveniment și redirecționare către activitatea ForgotPassword

În activitatea ce permite resetarea parolei, utilizatorul trebuie să își introducă email-ul pe care l-a setat atunci când a fost creat contul. Dacă email-ul corespunde cu cel din baza de date, se poate trata evenimentul de resetare a parolei, prin realizarea evenimentului de setOnClickListener() pe butonul „RESET PASSWORD”. Folosind acest eveniment, disponibil în figura 3.2.11, se apelează metoda pusă la dispoziție de librăria FirebaseAuth și anume *“sendPasswordResetEmail”.* Această metodă primește ca parametru email-ul introdus de utilizator în controlul de tip edit text. Dacă evenimentul se realizează cu succes, atunci cererea este procesată, iar utilizatorul va primi un email de resetare a parolei.

Firebase pune la dispoziție un șablon de trimitere a email-ului, care poate fi modificat de dezvoltator. Email-urile de resetare a parolei sunt trimite de pe o adresă de tip noreply@licenta-veche.firebaseapp.com, ce conțin un link de resetare. Utilizatorul își introduce noua parolă, iar aceasta se va actualiza în timp real în baza de date. Un model de email trimis se poate observa în anexa 10.

```
auth.sendPasswordResetEmail(emailReset).addOnCompleteListener((task) → {
        if (task.isSuccessful()) {
            Toast.makeText(ForgotPassword.this, "We have sent you instructions to reset your password!",
        } else {
            Toast.makeText(ForgotPassword.this, "Failed to send reset email!", Toast.LENGTH_SHORT).show()
        }
        progressDialog.dismiss();
});
```

Figura 3.2.11- Metoda apelată de resetare a parolei

Asemănător acțiunilor de logare și creare de cont au loc și modificările asupra contului. Utilizatorul își poate modifica datele din opțiunea meniului denumită PageMyAccount. Acesta poate să își modifice fie email-ul, fie parola sau pe amândouă în același timp. Pentru realizarea ambelor acțiuni se va identifica mai întâi utilizatorul curent, prin variabila user, care preia instanţa utilizatorului curent.

Acţiunea de modificare a email-ului se relizează prin completarea câmpului “New Email” cu noua adresă dorită. Prin metoda *updateEmail* pusă la dispoziție de librăria FirebaseAuth se primește ca parametru noul email introdus și se actualizează în baza de date. Această acțiune se realizează prin evenimentul de click asupra butonului CHANGE.

Acţiunea de modificare a parolei se realiezează tot prin intermediul butonului CHANGE, dar prin metoda *updatePassword*. Metoda primeşte ca parametru noua valoare introdusă în câmpul „New password”. Dacă nu intervine nici o eroare, datele vor fi actualizate în baza de date. În anexa 11 se poate observa implementarea ambelor metode. Antetul ambelor metode este reprezentat în figura 3.2.12.

```
user.updateEmail(txtChangeEmail.getText().toString().trim()).addOnCompleteListener(new OnCompleteListener<Void>
});

user.updatePassword(txtChangePassword.getText().toString().trim()).addOnCompleteListener((task) → {
```

Figura 3.2.12- Antetul metodelor updatePassword și updateEmail

Totuși, pentru a se realiza cu succes aceste modificări utilizatorul trebuie să aibă o activitate recentă în cont. Altfel, se vor generea erori și datele nu vor putea fi procesate. După ce datele sunt actualizate, utilizatorul este nevoit să se delogheze și să se autentifice cu noile date, deoarece re-autentificarea automată nu este posibilă.

Încheierea sesiunii curente se realizează prin evenimentul de click asupra butonului LOGOUT și prin apelarea metodei *signOut*, care încheie sesiunea. La momentul delogării, utilizatorul va primi un mesaj, de forma unui AlertDialog, şi va fi întrebat dacă doreşte sau nu să părăsească contul. Dacă va alege opţiunea „yes”, atunci utilizatorul va fi redirecţionat către MainActivity, pentru a realiza o nouă autentificare. În caz contrar, el va râmâne în contexul curent al aplicaţiei. Implementarea delogării se poate observa în figura 3.2.13.

```
btnLogout.setOnClickListener((v) → {
        firebaseAuth = FirebaseAuth.getInstance();
        FirebaseUser user = firebaseAuth.getCurrentUser();

        final AlertDialog.Builder builder = new AlertDialog.Builder(getActivity())
        builder.setMessage("Do you want to logout ?");
        builder.setPositiveButton("Yes!", (dialogInterface, i) → {

                firebaseAuth.signOut();
                getActivity().finish();
                Intent intent = new Intent(getActivity(), MainActivity.class);
                startActivity(intent);
        });

        builder.setNegativeButton("Not now", (dialogInterface, i) → {

                dialogInterface.dismiss();
        });

        AlertDialog dialog = builder.create();
        dialog.show();
```

Figura 3.2.13- Implementarea delogării

Trimiterea feedback-ului către dezvoltator se realizează din opțiunea de meniu PageFeedback. Aceasta pune la dispoziție un control de tip edit text în care se va scrie mesajul și un buton pentru trimiterea mesajului. Evenimentul de trimitere se realizează tot la acțiunea pe buton, folosind metoda setOnClickListener(). După cum se observă în figura 3.2.14, inițial se creează o instanță de Intent, care folosește acțiunea action_send ce specifică telefonului că se vor folosi serviciile de email instalate pe acesta.

Prin metoda *putExtra(Intent.EXTRA_EMAIL)* se reține un String cu adresa de email către care se va trimite mesajul. A doua metodă putExtra preia textul scris în control, iar împreună cu acțiunea action_send, lucrează pentru a trasnmite datele. Prin metoda *setType* se specifică tipul de email trimis. Transmiterea mesajului se realizează prin trimiterea Intent-ului ca parametru în metoda createChooser. Astfel, i se oferă utilizatorului posibilitatea de a alege prin ce aplicație să trimită feedback-ul.

```
send.setOnClickListener((v) → {
        to = (EditText) rootView.findViewById(R.id.editTextTo);
        mess = (EditText) rootView.findViewById(R.id.editTextFeedback);

        Intent email = new Intent(Intent.ACTION_SEND);
        email.putExtra(Intent.EXTRA_EMAIL, new String[]{to.getText().toString()});
        email.putExtra(Intent.EXTRA_TEXT, mess.getText().toString());
        email.setType("message/rfc822");
        startActivity(Intent.createChooser(email, "Choose app to send"));
```

Figura 3.2.14- Trimiterea de feedback prin email

Generarea întrebărilor din baza de date reprezintă cea mai importantă parte a aplicației. Pe baza unui formular completat în prealabil de către utilizator din opțiunea de meniu PageForm, se vor genera întrebările pe baza a două câmpuri: limbajul de programare cerut și tipul de job pentru care se aplică.

Inițial, aceste variabile introduse în edit text-uri sunt reținute local și partajate între activități folosind Intenturi. Metoda putExtra primește ca parametri o cheie unică și textul introdus de utilizator.

```
Intent i = new Intent(getContext(), ListaIntrebari.class);
i.putExtra("internship", level.getText().toString());
i.putExtra("junior", level.getText().toString());
```

Figura 3.2.15- Exemplu metoda putExtra

Auxiliar, se creează o clasă CustomAdapter în activitatea ListăÎntrebări, ce va fi folosită ca un holder pentru datele preluate din baza de date. Metodele suprascrise odată cu crearea acestei clase se vor folosi pentru a citi și scrie date din baza de date.

Elementele cu care interacționează utilizatorul în această activitate sunt susținute de un widget numit RecyclerView. Acest tip de widget, împreună cu un CardView, ajută la crearea de liste personalizate complexe. RecyclerView este mai evoluat decât un ListView, fiind un container folosit pentru a afișa seturi de date foarte mari și care se actualizează în timp real.[10]

Pentru a folosi un RecyclerView trebuie mai întâi să se specifice un adapter și un LayoutManager. Adapter-ul se creează prin extinderea clasei RecyclerView.Adapter și prin suprascrierea metodelor din această clasă. LayoutManager-ul folosit în aplicația curentă de adapter este de tip LinearLayoutManager, ceea ce înseamnă că elementele sunt afișate orizontal sau vertical și că se poate face scroll în listă. În anexa 12 se poate observa cum au fost instanțiate elementele RecyclerView și LinearLayoutManager în metoda onCreate().

Preluarea elementelor din baza de date se face în metoda updateList(). În această metodă se creează un obiect al clasei Bundle prin care se primesc în activitatea curentă valorile din edit text-urile din PageForm. Apoi se rețin în String-uri valorile introduse de la tastatură.

```
Bundle extras = getIntent().getExtras();
if(extras!=null) {

    String l1 = extras.getString("internship");
    String l2 = extras.getString("junior");
    String l3 = extras.getString("middle");
```

Figura 3.2.16- Exemplu de preluare a datelor din PageForm

După ce aceste valori au fost transmise între activități, se vor scrie condiții pentru fiecare caz în parte.

```
if (l1.contains("internship") && lg1.contains("C/C++")) {
    ref = database.getReference("CIntern");
    ref.addChildEventListener(new ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {

            result.add(dataSnapshot.getValue(Intrebare.class));
            adapter.notifyDataSetChanged();
        }
```

Figura 3.2.17- Exemplu de preluare a datelor din PageForm

În figura 3.2.17 se prezintă un exemplu al cazului în care utilizatorul alege din formular limbajul C/C++ și postul ca fiind un internship. Astfel, String-urile l1 și l2 verifică dacă valorile introduse de utilizator corespund cu valorile date ca parametru.

[10] https://developer.android.com/training/material/lists-cards.html

Dacă acestea corespund, următorul pas îl reprezintă citirea din baza de date. Prin variabila ref se identifică instanța curentă ce va fi folosită, în acest caz nodul CIntern. Evenimentul *addChildEventListener()* este apelat o singură dată pentru fiecare copil al nodului părinte și apoi de fiecare data când este adăugat un nou obiect la nodul copil. Evenimentul primește noul obiect într-o variabilă de tip DataSnapshot. Prin această variabilă se vor prelua valorile din baza de date și se vor atașa în lista customizată prin adapter. Pentru fiecare modificare asupra adapter-ului se va apela metoda notifyDataChanged(), pentru a anunța că s-a produs o modificare.

După cum am precizat în capitolele anterioare, în această activitate utilizatorul are opțiunea se a își alege întrebările favorite și de a le salva într-o altă activitate, denumită simbolic Favorite. Selectarea unei întrebări preferate se realizează printr-un eveniment asupra unei imagini în formă de inimă. Daca întrebarea este selectată, inima își va schimba culoarea în roșu, iar deselectarea întrebării va duce la schimbarea inimii intr-una transparentă.

Salvarea întrebărilor favorite în baza de date se va face prin intermediul nodului părinte Likes, după cum este prezentat în figura 3.2.18.

```
if (dataSnapshot.child(post_key).hasChild(cheie)) {

    likes.child(post_key).child(cheie).removeValue();
    processLike = false;
} else {
    likes.child(post_key).child(cheie).child("userId").setValue(auth.getCurrentUser().getUid());
    likes.child(post_key).child(cheie).child("intrebare").setValue(holder.txtIntrebare.getText());
    likes.child(post_key).child(cheie).child("raspuns").setValue(holder.txtRaspuns.getText());
    processLike = false;
}
```

Figura 3.2.18- Salvarea datelor în nodul Likes

În două variabile de tip String, post_key și cheie, se rețin valorile către nodurile copil care vor fi adăgate.

```
final String post_key=ref.getRef().getKey();
final String cheie=database.getReference(String.valueOf(position)).getKey();
```

Figura 3.2.19- Inițializarea obiectelor de tip String

În acest caz, *post_key* se va referi la nodul părinte de care aparțin întrebările care au fost setate ca favorile, iar *cheie* se va referi la poziția întrebării din cadrul adapter-ului. Salvarea în baza de date se face prin parcurgerea fiecărui nod copil în parte, prin metoda *child()* și prin setarea obiectelor sub formă de perechi chei-valoare, unde cheia este dată ca parametru în ultimul nod copil, iar valoarea este preluată din adapter, prin intermediul unui holder. Un exemplu al datelor din nodul Likes este disponibil în anexa 14.

Eliminarea unei întrebări setată ca favorită din baza de date se realizează prin metoda removeValue(). Se va determina poziția întrebării în adapter și se va elimina atât din baza de date cât și din RecyclerView.

Schimbarea culorii imaginii care arată dacă o întrebare este preferată sau nu se face prin intermediul metodei setLikeBtn(), care primește ca parametru String-ul cheie menționat anterior.

```
public void setLikeBtn(final String cheie){

    final String post_key=ref.getRef().getKey();
    likes.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (dataSnapshot.child(post_key).hasChild(cheie)){
                imgBtn.setImageResource(R.drawable.like1);
            }
            else{
                imgBtn.setImageResource(R.drawable.unlike1);
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
```

Figura 3.2.20- Metoda setLikeBtn()

În această metodă se verifică dacă nodul copil al obiectului de tip DataSnapshot are un alt copil, prin metoda *hasChild()*: în caz afirmativ, culoarea imaginii va fi setată ca roșu prin metoda *setImageResource*, iar în caz contrar culoarea va fi setată ca transparentă. Resursele pentru cele două imagini vor fi preluate din folder-ul /res/drawable.

Activitatea pentru afișarea întrebărilor favorite este asemănătoare cu activitatea ListăÎntrebări. Ea poate fi accesată de utilizator dintr-un meniu contextual. În cadrul acesteia se creează din nou o clasă CustomFavorite care extinde RecyclerView.Adapter și cu ajutorul căreia se va crea lista customizată. Citirea datelor se va face tot într-o metodă numită updateList(), în care se va genera referința către nodul părinte ce va fi folosit și prin care se va identifica utilizatorul curent după id-ul unic.

```
likes= database.getReference().child("Likes").child(auth.getCurrentUser().getUid());
```

Figura 3.2.21- Referința către instanța curentă a bazei de date

Preluarea datelor se face folosind un obiect de tip DataSnapshot, care prin metoda *getChildren()* reține toți copiii nodului părinte și realizează afișarea în adapter prin metoda *getValue()*. Dacă au loc modificări la nivel de listă, aceasta se va actualiza prin apelarea metodei *notifyDataSetChanged()*.

Pentru generarea quiz-urilor utilizatorul se va folosi de interfața PageQuiz. Aceasta pune la dispoziție diferite butoane din care utilizatorul poate să aleagă din ce limbaj de programare se vor genera întrebările. Momentan aplicația pune la dispoziție câte cinci întrebări pentru fiecare quiz, dar pe măsură ce se va dezolta baza de date, numărul întrebărilor va crește.

Pentru fiecare quiz, utilizatorul va vizualiza întrebarea într-un control de tip text view. Răspunsurile întrebărilor vor fi selectate din controale de tip radio-button, deoarece răspunsul va fi unic. Accesarea următoarei întrebări se realizează printr-un eveniment de click pe butonul NEXT. O dată cu trecerea la următoarea întrebare răspunsul selectat va fi salvat local, folosind clasa SharedPreferences. În activitate, datele sunt preluate dintr-o listă de string-uri, iar scorul inițial este 0. Varianta de răspuns este preluată din RadioButton prin metoda getText(), iar dacă aceasta corespunde cu răspunsul corect din vectorul de string-uri atunci se va trece la următoarea întrebare și scorul va crește. În caz contrar, se va afișa un mesaj care va atenționa utilizatorul că răspunsul selectat este greșit. La finalul quiz-ului scorul este reținut într-un text view și salvat local.

```
btnNext.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            RadioGroup grp=(RadioGroup)findViewById(R.id.radioGroup1);
            RadioButton answer=(RadioButton)findViewById(grp.getCheckedRadioButtonId());
            if(currentQ.getANSWER().equals(answer.getText()))
            {
                score++;
            }
            currentQ=quesList.get(qid);
            setQuestionView();
        }
    });
```

Figura 3.2.22- Evenimentul de click pentru butonul Next

Tot din această activitate, utilizatorul poate să își urmărească evoluția în timp printr-un grafic. Accesarea graficului se face printr-un eveniment de click pe text view-ul “See chart evolution”. Implementarea graficului se realizează folosind librăria MP Android Chart Library, iar graficul va fi de tip group și va conține o animație. Importarea librăriei se face prin adăugarea ei ca dependință în directorul build.gradle.

```
compile 'com.github.PhilJay:MPAndroidChart:v2.0.9'
```

Figura 3.2.23- Importatea librăriei ca dependință

Layout-ul folosit pentru desenarea graficului este de tip RelativeLayout. Scorurile sunt reținute într-o listă de variabile float, iar adăugarea lor se face prin metoda *add*(). Desenarea fiecărei bare din grafic se face folosind o instanță a clasei BarEntry, care primește ca parametri un element din listă și poziția pe care va fi afișată bara.

Datele sunt setate pe grafic folosind clasa un obiect din clasa BarData și metoda setData(), care primește ca paramtru lista de obiecte de tip BarEntry. Animația este posibilă folosind metoda *setAnimation()*.

```
BarData data = new BarData(getXAxisValues(), getDataSet());
chart.setData(data);
chart.setDescription("My Chart");
chart.animateXY(2000, 2000);
chart.invalidate();
```

Figura 3.2.24- Setarea datelor pe grafic

3.3 PROBLEME TEHNICE ȘI SOLUȚII GĂSITE

În ceea ce privește problemele întâmpinate la implementarea aplicației, cele mai multe au fost în legătură cu utilizarea bazei de date. Întrucât s-a utilizat o bază de date nerelațională, încă de la început a fost nevoie ca datele să fie structurate corect, pentru a facilita manipularea acestora. În ciuda acestui lucru, citirea și scrierea datelor au fost problematice, deoarece interogările se realizau diferit față de cum se procedează cu o bază de date relațională. Aceste probleme duceau astfel la erori de compilare, precum și la afișarea greșită sau deloc a rezultatelor. Rezolvarea a presupus foarte multă documentare despre gestiunea bazelor de date NoSql, precum și urmărirea tutorialelor pentru a implementa corect anumite metode.

O altă problemă întâmpinată a fost apariția de bug-uri la nivel de fragmente. Deaorece la nivel de liste customizate s-au folosit anumite controale pentru afișarea datelor, de tip RecyclerView sau CardView, acestea au dus la apariția unor incompatibilități când erau utilizate într-un fragment. Soluția pentru această problemă a fost implementarea unor funcționalități în activități, în loc de fragmente, pentru a se elimina bug-urile.

Probleme s-au regăsit și la nivel de design și layout-uri. Rularea pe un dispozitiv mobil a aplicației a dus la modificarea poziției layout-ului, precum și la modificarea pozițiilor controalelor. Adaptarea design-ului la unul potrivit pentru telefonul mobil a dus la rezolvarea acestei probleme.

O altă problemă întâmpinată a fost trimiterea feedback-ului către dezvoltator. Inițial mail-ul trebuia trimis în background, folosind JavaMailApi, fără ca utilizatorul să mai deschidă aplicațiile specifice. Acest lucru nu a fost posibil, deoarece aplicația nu putea să se conecteze la serverul SMTP. Din această cauză s-a ales trimiterea feedback-ului folosind aplicațiile YahooMail sau Gmail.

4. CONCLUZII

Sintetizând cele prezentate anterior, lucrarea de față își propune să vină în ajutor tuturor celor care îsi caută un loc de muncă în domeniul IT, printr-o aplicație mobilă ușor de utilizat și la îndemâna oricărui utilizator.

Aplicația pune la dispoziție o interfață ce permite utilizatorilor să parcurgă seturi de întrebări tehnice din diferite limbaje de programare, în funcție de tipul de job pentru care aplică și de limbajul cerut. Prin acest lucru aplicația aduce o noutate față de aplicațiile deja existente pe piață, deoarece oferă un mod de filtrare și de personalizare al rezultatelor prin completarea unui formular. În plus, ea permite utilizatorilor să își testeze cunoștiințele prin rezolvarea unor quiz-uri în funcție de limbajul de programare dorit sau quiz-uri mixte, cu întrebări din mai multe limbaje. O noutate mai este și faptul că trecerea la următoarea întrebare este condiționată de răpsunsul corect la precedenta, spre deosebire de aplicațiile în care răspunsurile corecte sunt aflate de abia la finalul quiz-ului, o dată cu punctajul. În plus, utilizatorul își poate ține o evidență a punctajelor printr-un grafic care stochează automat toate rezultatele și care determină evoluția nivelului de cunoștiințe.

O altă funcționalitate a aplicației o reprezintă opțiunea de a seta ca favorite întrebările dorite și vizualizarea lor într-o activitate specială. Această opțiune nu aduce o noutate, deoarece este implementată în foarte multe aplicații de pe piață, dar este un mod bun de gestionare a preferințelor utilizatorului. Utilizarea aplicației este condiționată de un sistem de login, ce se realizează prin introducerea unei adrese de email și a unei parole. Dacă utilizatorul nu are un cont deja existent el își poate crea unul, prin completarea unor cămpuri specifice, iar apoi are loc înregistrarea automată. Această funcționalitate este necesară pentru a păstra o evidență a tuturor celor care folosesc aplicația, precum și modul în care o folosesc. În plus, utilizatorii au opțiunea de a își modifica datele contului, prin actualizarea acestora dintr-o activitate specifică.

O ultimă funcționalitate o reprezintă trimiterea de feedback dezvoltatorului prin intermediul email-ului, pentru a primi sugestii de îmbunătățire sau eventualele probleme legate de aplicație.

Toate aceste funcționalități sunt ușor de utilizat datorită interfeței user-friendly și a meniului contextual, fără a fi nevoie de un ghid de utilizare al aplicației. Scopul acesteia nu este numai de a ajuta candidații în obținerea unui job, ci și de a oferi un mediu de învățare celor aflați la început de drum, care vor să învețe un limbaj de programare. Chiar dacă unele din funcționalități pot fi regăsite separat în aplicațiile de pe piață, faptul că ele sunt reunite într-un

singur loc aduce un plus, fără ca utilizatorul să fie nevoit să descarce câte o aplicație diferită pentru fiecare modul în parte.

Totuși, aplicația prezintă și anumite limitări, din punct de vedere tehnic. De exemplu, o limitare destul de importantă este aceea că aplicația este disponibilă doar pe telefoane mobile, iar pe tablete nu. Aceste limitări nu sunt majore și nu influențează funcționalitatea. Astfel, în viitor se poate modifica aplicația, pentru a permite îmbunătățiri. În plus, se pot face modificări la nivel de interfață, prin utilizarea framework-ului JQuery mobile. În viitor aplicația poate fi dezvoltată să funcționeze și pe alte dispozitive, eventual și folosind sistemul de operare IOS.

Totodată, se mai pot adăuga module aplicației, cum ar fi: setarea unei poze de profil pentru contul utilizatorului; permiterea logării prin diferite conturi: Facebook, Twitter sau Google; adăugarea unor noi limbaje de programare este posibilă, precum și a mai multor întrebări și quiz-uri. Mai este posibilă adăugarea unei opțiuni de partajare a rezultatelor între utilizatori.

Un aspect ce mai poate fi implementat este migrarea bazei de date, dintr-o bază de date nerelațională într-una relațională. Acest lucru ar ușura foarte mult accesul la date, deoarece interogările folosind baze de date NoSql pot fi dificile.

În concluzie, consider că aplicația la momentul actual întrunește cele necesare pentru scopul propus. Ea reprezintă o bază pentru orice candidat care își dorește să reușească în domeniul IT prin funcționalitățile puse la dispoziție.