**MINISTERUL EDUCAŢIEI AL REPUBLICII MOLDOVA**

**UNIVERSITATEA DE STAT „ALECU RUSSO” DIN BĂLŢI**

**FACULTATEA DE ŞTIINŢE REALE, ECONOMICE ȘI ALE MEDIULUI**

**CATEDRA DE MATEMATICĂ ȘI INFORMATICĂ**

**ANALIZA COMPARATIVĂ A LIMBAJELOR JAVA ȘI KOTLIN CA INSTRUMENT DE DEZVOLTARE A APLICAȚIILOR MOBILE**

**TEZĂ DE AN**

**Autor:**

Student al grupei IS21Z

**Vitalie SANDIUC**

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

*(semnătura)*

**Conducător științific:**

**Sergiu CHILAT**

Lector univ.

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

*(semnătura)*

**BĂLȚI, 2021**

**CUPRINS**

[INTRODUCERE 3](#_Toc72345212)

[1. DESCRIEREA LIMBAJELOR DE PROGRAMARE KOTLIN ȘI JAVA 4](#_Toc72345213)

[1.1. Limbajul de programare Java pentru elaborarea aplicațiilor mobile 4](#_Toc72345214)

[1.2. Avantajele utilizării limbajului de programare Java pentru elaborarea aplicațiilor mobile 4](#_Toc72345215)

[1.3. Dezavantajele utilizării limbajului de programare Java pentru elaborarea aplicațiilor mobile 5](#_Toc72345216)

[1.4. Concluzia utilizării limbajului Java pentru elaborarea aplicațiilor mobile 7](#_Toc72345217)

[1.5. Limbajul de programare Kotlin pentru elaborarea aplicațiilor mobile 7](#_Toc72345218)

[1.6. Avantajele utilizării limbajului de programare Kotlin pentru elaborarea aplicațiilor mobile 7](#_Toc72345219)

[1.7. Dezavantajele utilizării limbajului de programare Kotlin pentru elaborarea aplicațiilor mobile 11](#_Toc72345220)

[1.8. Concluzia utilizării limbajului Kotlin pentru elaborarea aplicațiilor mobile 11](#_Toc72345221)

[2. ANALIZA COMPARATIVĂ A SINTAXEI JAVA ȘI KOTLIN 13](#_Toc72345222)

[3. EXEMPLU PRACTIC DE APLICAȚIE 17](#_Toc72345223)

[3.1. Introducere 17](#_Toc72345224)

[3.2. Descrierea aplicației 17](#_Toc72345225)

[3.3. Elaborarea aplicației în limbajul de programare Kotlin 17](#_Toc72345226)

[3.4. Elaborarea aplicației în limbajul de programare Java 25](#_Toc72345227)

[CONCLUZII 27](#_Toc72345228)

[BIBLIOGRAFIE 28](#_Toc72345229)

INTRODUCERE

Odată cu progresul sferei IT în lumea cotidiană, dar și cu apariția rapidă a site-urilor web, aplicațiilor atât mobile, cât și desktop, lumea pășește într-o etapă nouă a dezvoltării. Cu orice progres apare dorința de a oferi cel mai bun product și de a atinge succesul, astfel tot mai multe persoane leagă viața lor cu domeniul IT.

*Actualitatea studiului:* În urma analizei pieței, dar și a informației din diferite surse, odată cu apariția mai multor site-uri, dar și a device-urilor tot mai performante, și anume telefoanele mobile, crește și necesitatea de a oferi utilizatorilor aplicații performante, cu un funcțional amplu, dar și diferit față de concurenți.

*Scopul lucrării* - Cercetarea limbajelor de programare java și kotlin ca instrument de elaborare a aplicațiilor mobile și elaborarea unei aplicații mobile folosind aceste limbaje.

Pentru atingerea scopului, au fost puse următoarele sarcini:

* Studierea limbajului de programare Java;
* Studierea limbajului de programare Kotlin;
* Studierea și analiza diferențelor dintre cele două;
* Elaborarea aplicației mobile atât prin limbajul de programare Java cât și prin Kotlin cu aplicarea instrumentelor studiate.

*Valoarea teoretică:* partea teoretică a lucrării date este importantă, pentru că au fost studiate două limbaje de programare Java și Kotlin și analizate diferențele dintre ele.

*Valoarea practică* constă din elaborarea aplicațiilor pe limbajele de programare Kotlin și Java și reprezintă valoare pentru programatorii aplicațiilor mobile sau pentru studenți.

Lucrarea este structurată în 3 capitole. În primul capitol sunt prezentate avantajele și dezavantajele fiecărui limbaj.

În capitolul doi este prezentată diferența de sintaxă dintre cele două limbaje de programare.

În capitolul trei este prezentată aplicația elaborată prin ambele limbaje.

Teza include 28 pagini, 6 figuri, un tabel și o listă de 12 surse bibliografice.

1. DESCRIEREA LIMBAJELOR DE PROGRAMARE KOTLIN ȘI JAVA
   1. Limbajul de programare Java pentru elaborarea aplicațiilor mobile

Java a fost limbajul de programare implicit încă de la apariția primei platforme Android, fiind introdusă în anul 2008 ca limbaj de programare orientat pe obiect și elaborat de Sun Microsystems. Mai multe aplicații mobile au ca bază acest limbaj de programare (fig.1.1). Java este un limbaj care posedă atât avantaje cât și dezavantaje în cadrul utilizării sale. A fost dezvoltat pentru a fi independent de platforma pentru care are loc scrierea soft-ului, dar oferă o securitate aplicațiilor, având sloganul “Scris odată și folosit peste tot”[2]. Puține limbaje de programare se plasează pe un top de popularitate mai mult de 20 de ani, Java este unul dintre ele[2]. Pentru programarea în java pe platforma android este de ajuns un mediu de dezvoltare ca Android Studio sau IntelliJ IDEA și SDK-ul pentru compilarea aplicațiilor. Fiind un limbaj ușor, este predat și copiilor care doresc să învețe programarea, atât pe Android cât și pe alte platforme.

s

Fig. 1.1. Aplicații mobile populare elaborate pe Java.

* 1. Avantajele utilizării limbajului de programare Java pentru elaborarea aplicațiilor mobile
* Limbajul este prezentat ca fiind unul ușor de a fi învățat și studiat, fiind prietenos pentru începători;
* Oferă posibilitatea de cross-platform, dezvoltatorii Java pot crea aplicații atât pe Android, cât și pe iOS utilizând GraalVM, dar și aplicații desktop;
* AndroidSDK este bazat pe Java, conținând destul de multe biblioteci Java standarte;
* Limbajul Java are un ecosistem open-source larg;
* Aplicațiile mobile scrise pe Java sunt mai compacte față de limbajul Kotlin (din capitolul 3 aplicația, 2.0 MB în comparație cu 3.6 MB din Kotlin);
* Smartphone-urile al căror specificații tehnice sunt slabe, aplicațiile scrise pe Java tind să funcționeze mai rapid decât pe Kotlin;
* Este limbaj orientat pe obiect și oferă posibilitatea ușoară de a crea aplicații modulare;
* Multi-thread, oferă posibilitatea îndeplinirea mai multor task-uri în același timp;
* Oferă cod gata de a fi folosit în proiectele dorite[1];
* Java este ușor de a fi folosit, de a compila și debugging-ul, dar și deploy-ul este simplu;
* Dezvoltatorii cu experiență în limbajul dat, pot ușor deveni dezvoltatori de aplicații pe platforma Android folosind limbajul de programare Java;
* Comunitate mare, biblioteci multe, module gata făcute, pentru rezolvarea unei probleme nu necesită mult timp pentru căutarea soluției, în comparație cu Kotlin;
* Include utilizarea excepțiilor ce permit detectarea, dar și rezolvarea erorilor mai ușor;
* Suportă conversiile implicite;
* Folosește membrii statici;
* Oferă operatorul ternar.
  1. Dezavantajele utilizării limbajului de programare Java pentru elaborarea aplicațiilor mobile
* Sintaxa folosită de Java este mai complicată în comparație cu Kotlin;
* Câteodată Java cauzează probleme cu designul API Android;
* Aplicațiile mobile scrise pe Java necesită mai mult cod în comparație cu cele scrise pe Kotlin, iar codul mai mult duce la generarea mai rapidă a bug-urilor, dar și a erorilor

Pentru adresarea la un TextView în Java este următorul cod:

*TextView myTextView = (TextView) findViewById(R.id.text\_my\_id);*

*myTextView.setText(“My Text”);*

în limbajul de programare Kotlin arată în felul dat:

*myTextView.text = “My Text”*

fiind mult mai simplu și accesibil de înțeles;

* Cu trecerea timpului, întreținerea codului pe Java creează tot mai multe dificultăți, și mai devreme sau mai târziu, Kotlin va fi un limbaj de programare pe care va apărea dorința de a trece;
* Pentru accesarea unui component din GUI este necesară crearea unei variabile ce privește tipul componentei, este necesară folosirea findViewBy id pentru găsirea în GUI și apoi abia pot fi făcute operații asupra ei;
* Consumă foarte mult timp pentru a fi învățat, dar și amploarea limbajului este foarte mare;
* Necesită foarte multă memorie, și din acest motiv, aplicațiile, pot rula greu câteodată;
* Cere mult lucru manual, ce crește numărul de erori potențiale ce pot apărea;
* Nu are funcții extensii;
* Siguranța nulă, Java oferă posibilitatea de a atribui valoarea zero la obiecte, dacă dezvoltatorul o va face, imediat va primi un “NullPointerException”, ce îngreunează procesul de elaborare a aplicației;
* La efectuarea unei operații în Java, este strict necesar verificarea variabilelor și tipul lor;
* Este limitată la programarea orientată pe obiect;
* Java nu oferă funcțiile inline;
* Nu putem schimba ordinea argumenților în funcție;
* Lipsesc valorile implicite în declararea funcțiilor, dar și a claselor;

Spre exemplu în Kotlin : *fun myFunction(name: String = ”dasda**”) ,* unde funcția poate fi apelată fără argumente, dar și clasa are așa posibilitate, în java însă așa ceva nu există, nici în funcții, nici în clase.

* Getter și setter în Java sunt necesari de a fi scriși manual, iar atunci când clasa prezintă mai multe proprietăți devine o adevărată probleme de a fi scrise, chiar dacă unele medii de programare oferă generarea automată a lor, întreținerea codului și lungimea lui se transformă în sute de rânduri în plus, în Kotlin sunt prezenți implicit, pot fi construiți după necesitate pentru modificarea lor.
  1. Concluzia utilizării limbajului Java pentru elaborarea aplicațiilor mobile

Atunci când este vorba despre dezvoltarea aplicațiilor mobile, o majoritate aleg utilizarea limbajului Java, deoarece însăși platforma Android a fost elaborată în Java, de asemenea și faptul că este un limbaj de programare orientat pe obiect, folosit de aproape 20 ani, fiind unul dintre al doilea cel mai activ limbaj de programare de pe Github. Astfel, limbajul dat se consideră unul important de a fi utilizat și învățat pentru elaborarea aplicațiilor mobile pe platforma Android, atât pentru faptul că oferă aproape tot ce este necesar unui dezvoltator, dar cât și pentru actualitatea lui până și în zilele de astăzi în companiile mari ale lumii.

* 1. Limbajul de programare Kotlin pentru elaborarea aplicațiilor mobile

Kotlin este un limbaj cross-platformă, tipizat static, elaborat pentru a interopera cu limbajul de programare Java, iar versiunea de JVM (Java Virtual Machine), ce permite unui calculator de a rula aplicații Java, depinde de el. Kotlin reprezintă un limbaj nou de programare, care ușor s-a extins în lume, începe a domina platforma Android și tot mai multe aplicații încep a fi elaborate anume pe acest limbaj. Limbajul este folosit pe mai multe platforme, cum ar fi Android, JVM, iOS, macOS, watchOS, tvOS, Windows, Linux, WebAssembly și altele. Din 2019 Google a început a prefera limbajul Kotlin pentru dezvoltarea aplicațiilor pe Android, ceea ce a conferit trecerea pe el a multor dezvoltatori de aplicații mobile.

* 1. Avantajele utilizării limbajului de programare Kotlin pentru elaborarea aplicațiilor mobile
* Ca limbaj nou, se dezvoltă foarte rapid, iar pe plan de viitor Kotlin reprezintă un limbaj cu o perspectivă mult mai înaltă, decât Java;
* Dezvoltatorii de aplicații mobile pe Java, cu ușurință pot trece la Kotlin, fiind ușor de învățat;
* Limbaj compact și dinamic, necesită puțin timp de a scrie codul;
* Limbajul de programare Kotlin și Swift, au multe lucruri în comun, ce conferă un plus pentru orizonturile dezvoltatorului;
* Calitatea codului pe Kotlin mereu este la un nivel mai înalt, față de Java;
* Cantitatea codului este redusă mereu;
* Prezența tipurilor de variabile *lateinit*, ce simplifică mai mult lucrul cu diferite obiecte;
* Codul pe Kotlin este mai citabil decât cel pe Java;
* Este compatibil cu toate librăriile și framework-urile Java;
* Clasele data, în Kotlin sunt mult mai comode de a efectua lucrul cu ele, decât în Java, spre exemplu în Kotlin:

Este creată clasa de tip data, ce are în constructor nume, vârsta, grupa și adresa reprezentarea mai jos, după care pentru a opera cu această listă, este de ajuns următoarele linii:

*data class Students (*

*val name: String,*

*val age: Int,*

*val group: String,*

*val address:String*

*)*

*fun main() {*

*val DB: ArrayList<Students> = ArrayList()*

*DB.add(0, Students(“Ion”, 20, “IS21Z”, “Stefan cel Mare 25”))*

*DB.add(1, Students(“Andrei”, 21, “IS21Z”, “Stefan cel Mare 22”))*

*DB.forEach{*

*println(it.name)*

*println(it.age)*

*println(it.group)*

*println(it.address) }*

Acest exemplu în limbajul java, observă un cod considerabil mai voluminos, același cod, cu aceeași funcționalitate în Java arată în felul următor:

*Main.java*

*ArrayList<Students> DB = new ArrayList<>();*

*DB.add(0, new Students(“Andrei”, 19, “IS21Z”, “Stefan cel Mare 25”));*

*DB.add(0, new Students(“Ion”, 20, “IS21Z”, “Stefan cel Mare 23”));*

*DB.forEach(listItem -> {*

*System.out.println(listItem.getName());*

*System.out.println(listItem.getAge());*

*System.out.println(listItem.getGroup());*

*System.out.println(listItem.getAddress());*

*}*

*Students.java*

*public class Students {*

*private final String name;*

*private final int age;*

*private final String group;*

*private final String address;*

*public String getName() { return this.name}*

*public int getAge() { return this.age}*

*public String getGroup() { return this.group}*

*public String getAddress() {return this.address}*

*Students(String name, int age, String group, String address) {*

*this.name = name;*

*this.age = age;*

*this.group = group;*

*this.address = address;*

*}*

*}*

În Kotlin este de ajuns de a fi declarat doar constructorul la clasa data, și atât, restul necesarului este gata, adică getterii și setterii sunt implicit, la necesitate funcționalul lor poate fi modificat.

* Funcțiile extensie, în Java ele nu există, iar în Kotlin creează un aspect mult mai clar în folosire;
* Kotlin posedă Lambda expresii;
* Kotlin este protejat împotriva NullPointerException;
* Obiectele Singleton în Kotlin sunt mult mai ușoare de a fi create exemplu:

*object Singleton {*

*init {*

*println(“clasa Singleton”)*

*}*

*var myVariable = “Azi e vineri”*

*fun printMyVariable() {*

*println(myVariable)*

*}*

*}*

*fun main(args: Array<String>) {*

*Singleton.printMyVariable()*

*Singleton.myVariable = “Azi e luni”*

*var a = A()*

*}*

*class A {*

*init {*

*println(“Clasa A. Singleton myVariable : ${Singleton.myVariable}”)*

*Singleton.printMyVariable()*

*}*

*}*

În urma executării va fi afișat :

*clasa Singleton*

*Azi e vineri*

*Clasa A. Singleton myVariable :Azi e luni*

*Azi e luni*

Pe java însăși Singleton-ul arată astfel, fără implementarea funcționalului de mai sus, întrucât codul nu e citabil și cu prezență de termeni ca *synchronized*:

*public class Singleton {*

*private static Singleton myInstance = null;*

*private Singleton() {*

*}*

*public static Singleton getInstance() {*

*if (instance == null) {*

*synchronized (Singleton.class) {*

*if (myInstance == null) myInstance = new Singleton();*

*}*

*}*

*return myInstance;*

*}*

*}*

* Excepțiile sunt mai comode de a fi utilizate;
* În Kotlin switch-ul este schimbat cu “when”, care este mult mai comod decât switch-ul, deoarece în switch poate fi ușor uitat un break, un case, sau când se adaugă o nouă valoare enum și se uită actualizarea tuturor cazurilor din switch, la fel și when-ul poate fi folosit în loc de lanțurile if-else, care face mult mai ușoară urmărirea codului;
* Combină programarea funcțională și procedurală;
* Kotlin evită multe din categoriile defecte ce apar în limbajul Java;
* Null Pointer-ul, în comparație cu Java nulitatea este mult mai compactă de a fi verificată:

if (y == null) y.foo(); iar în Kotlin: y?.foo().

* 1. Dezavantajele utilizării limbajului de programare Kotlin pentru elaborarea aplicațiilor mobile
* Surse puține de a învăța Kotlin[1];
* Viteza de compilare este mai mică decât Java (din capitolul practic, java – 13 secunde, kotlin – 14 secunde, în proiecte mai mari aceste cifre diferă semnificativ);
* Kotlin nu dispune de membrii statici;
* Tipurile wildcard în Kotlin lipsesc, iar în Java sunt prezente;
* Absența tipurilor de date primitive, în Kotlin nu pot fi declarate direct, el folosește clase (Int, Float) ca un obiect ce împachetează primitivele;
* Lipsa operatorului ternar “a < b ? a : b”, iar în Kotlin arată în modul dat:

minValue = if (a < b) a else b

* 1. Concluzia utilizării limbajului Kotlin pentru elaborarea aplicațiilor mobile

Fiind un limbaj nou, ce se dezvoltă rapid, Kotlin apare ca un limbaj pentru acoperirea minusurilor limbajului Java, în Kotlin există mult mai multe oportunități, fapt ce oferă dezvoltatorului un spectru larg de aplicare tehnologiilor lui. Totuși, Kotlin oferă niște tehnologii noi de dezvoltare a aplicațiilor mobile, dar oricum duce lipsa unor aspecte destul de importante. Kotlin oferă scrierea codului mai clară și citabilă, dar și utilizarea mai scurtă a unor metode, ceea ce îi oferă un plus destul de înalt. Ca limbaj nou, se dezvoltă destul de rapid, dar totuși nucleul dezvoltării aplicațiilor mobile la moment îl constituie limbajul de programare Java, iar până a ajunge la un rezultat dorit, Kotlin ca limbaj, trebuie dezvoltat, atât el, cât și comunitatea sa.

1. ANALIZA COMPARATIVĂ A SINTAXEI JAVA ȘI KOTLIN

Tabelul 1. Diferențele de sintaxă între Java și Kotlin

|  |  |
| --- | --- |
| Java | Kotlin |
| Declararea variabilelor | |
| Declararea variabilelor are loc prin utilizarea modificatorilor de acces în față, implicit dacă lipsește modificatorul de acces, atunci variabila se consideră publică, cuvintele cheie static final, urmat de tipul de date a variabilei și denumirea variabilei. | Declararea variabilelor are loc prin utilizarea modificatorilor de acces în față, lipsa modificatorului de acces setează implicit variabila publică, urmat de cuvântul cheie var sau val spațiu liber și numele denumirea variabilei, două puncte și tipul de date a variabilei. |
| Sintaxa și modalitate adresării către layout | |
| Pentru adresarea la anumite componente din layout (xml) este necesar de a fi scris View Binding-ul, astfel pentru a interpreta evenimentul de apăsare a unui buton este necesar:  *Button javaBtn = findViewById(R.id.javaBtn); javaBtn.setOnClickListener(new View.onCLickListener() {*  *@Override*  *public void onClick(View v) {*  *…*  *}*  *}* | Pentru adresarea la anumite componente din layout, în limbajul de programare Kotlin view binding-ul este deja prezent, astfel este de ajuns de a o adresare simplă către component:  *kotlinBtn.setOnClickListener {*  *…*  *}* |
| *Switch / when* | |
| Switch-ul arată în modul următor:  *switch(operation) {*  *case ‘+’: return a + b; break;*  *case ‘-‘ : return a – b; break;*  *default: …}* | Switch-ul nu există, este înlocuit cu construcția when:  *when (operation) {*  *‘+’ -> return a + b*  *‘-‘ -> return a – b*  *else -> ….}* |
| Expresii lambda | |
| Sintaxa expresiilor lambda arată în modul următor:  *parametru -> expresie* sau  *(parametru1, parametru2) -> {//cod}* | Sintaxa expresiilor lambda arată în modul următor:  *{ parametru1, parametru2 -> cod }* |
| Pentru manipulările cu valorile null se folosește  *if (myVariable != null) {*  *//cod*  *}* | Folosește mecanismul nul de siguranță, astfel:  *var name: String = “Ion”*  *name = null // va apărea eroare.*  Mecanismul nul:  *var name: String? = “Ion”*  *name = null // nu va apărea eroare* |
| Declararea claselor | |
| Variabilele globale:  *public class FirstClass {*  *public static int globalResult = 5;*  *}* | *class FirstClass {*  *companion object {*  *val globalResult = 5*  *}*  *}* |
| *Smart Cast* | |
|  | *fun smartCast(element: Any) {*  *if (element is String) {*  *print(element.length)*  *}*  *}* |
| Funcții extensie | |
|  | *fun Int.incrementByOne(): Int {*  *return this + 1;*  *}*  *var yourNumber = 6*  *var result = yourNumber.incrementByOne()*  *// printează 7.* |
| Moșternirea | |
| Are loc prin cuvântul *extends.*  *class Example extends ExampleInheritance* | *class Example : ExampleInheritance*  *open class ExampleInheritance* |
| Constante | |
| Constantele se declară ca final.  *final int yourNumber = 5;* | const val CONST\_EXAMPLE: Int = 5;  const val example: String = getExample();  fun getExample(): String {  return „example function’  } |
| Instanțierea claselor | |
| Pentru instanțierea clasei și inițializarea ei se folosește cuvântul cheie “new” | Pentru instanțierea și inițializarea clasei cuvântul new lipsește. |
| Variabile acționate mai târziu | |
| La utilizarea oarecărei construcții unde există necesitatea de o variabilă care va fi inițializată peste careva interval de timp, în Java este aplicat mecanismul cu sintaxa dată:  *Example example;*  unde Example reprezintă o clasă.  Și construcția uzuală din java:  *Example example = new Example();*  Obține forma următoare:  *example = new Example();* | Pentru declararea în Kotlin a unei variabile care va fi inițializată pe parcursul programului se foloesește:  *var example: TestExample? = null*  (unde *TestExample* este o clasă luată aleator)  Această construcție obligă de a face mereu verificări null în fiecare dată când folosim variabila *example*. În Java problema dată nu persistă, pentru că totul poate fi null, iar declararea unui câmp neinițializat nu este nimic deosebit. |
| Liste | |
| Listele se declară, necesită inițializate, necesită operații de adăugare a elementelor. | Listele sunt simple de a fi declarate, de a fi adăugate elemente și de a fi parcurse. |
| Instrucțiunea de decizie și diapazonul dintre cifre | |
| *int x = 5;*  *if (x >= 1 && x <=8) {*  *System.out.println(“Este in diapazon”);*  *}* | *Var x:Int = 5*  *if (x in 1...8) {*  *println(“Este in diapazon”)*  *}* |
| Funcțiile | |
| Funcțiile în Java se scriu în modul următor:  *int sum(int a, int b) {*  *return a + b;*  *}*  Unde int înainte de sum reprezintă tipul valorii pe care funcția trebuie să returneze. | *fun sum(a: Int, b: Int): Int {*  *return a + b*  *}*  Apare cuvântul *fun* , tipul returnat de funcție se notează *:Int.* |

Tabelul 1 observă diferențe dintre mecanismele de efectuare a oare căror operații, dar și sintaxa, fiind două limbaje de programare Kotlin și Java prezintă asemănări dintre sintaxă, dar și deosebiri.

Cele mai semnificative deosebiri sunt în declararea variabilelor în Java – *int myNumber = 5;* și în Kotlin *var myNumber: Int = 5*, se atestă lipsa *;* în Kotlin, dar oferind ambele același principiu de declarare, totuși o altă diferență dintre sintaxă, dar și logică la declararea variabilelor este că limbajul Kotlin utilizează o înfășurare a tipurilor de date față de Java unde sunt primitive. La fel și mecanismele null sunt modificate, tabelul 1 arată aplicarea acestor mecanisme, în Java pe tot parcursul programului sunt necesare verificări null, iar în Kotlin deja sunt făcute pentru dezvoltator. La moștenirea claselor, Java folosește cuvântul extends, iar după cum este realizat în tabel este necesar de a înțelege că în Kotlin orice clasă este finală, ea moștenește de la superclasa Any, iar pentru realizarea moștenirii dintre clase este necesară clasa care este moștenită să fie deschisă, utilizând cuvântul open.

Kotlin și Java prezintă aproximativ similară, dar logica și unele mecanisme complexe sunt mult mai ușor de a fi scrise pe Kotlin, dacă în Java este necesară declararea unei variabile care mai apoi să fie inițializată peste ceva timp de la lansarea programului se folosesc 2 linii de cod, iar în Kotlin este una singură ce oferă și un alt mecanism, și anume *lateinitvar,* pe care Java nu îl posedă. Chiar și verificarea unui număr dacă se află în diapazon este mai intuitivă pe Kotlin pentru cine citește pentru prima dată codul, decât pe Java. O altă diferență este între liste, spre exemplu pe Java o listă arată în modul următor: *ArrayList<Integer> numberList = new ArrayList<>();* după care urmează operații de adăugare, care iar necesită foarte mult cod, pe Kotlin e intuitiv și simplu: *val items = listOf<Int>(5,6,7,9)* Fiind luate în considerare cerințele față de cod, dar și mulți alți factori, limbajul de programare Kotlin apare ca unul mult mai scurt după numărul liniilor de cod, cu mecanisme mult mai simple și cu o apropiere mai citabilă a codului, astfel când este vorba despre acest aspect, Kotlin este învingător.

3. EXEMPLU PRACTIC DE APLICAȚIE

* 1. Introducere

Va fi realizat aceeași aplicație mobilă pe platforma Android, realizarea va avea loc în limbajul de programare Java și Kotlin, acest fapt rezultă la prezentarea diferențelor dintre aceste două limbaje pentru realizarea aplicațiilor mobile.

* 1. Descrierea aplicației

Exemplu de aplicație ce va oferi diferențele dintre limbajele date, este aplicație meteo, ce va face un request la un API și va obține datele, afișându-le pe ecran.

* 1. Elaborarea aplicației în limbajul de programare Kotlin

Se începe cu crearea unui proiect gol, utilizând API level 21.

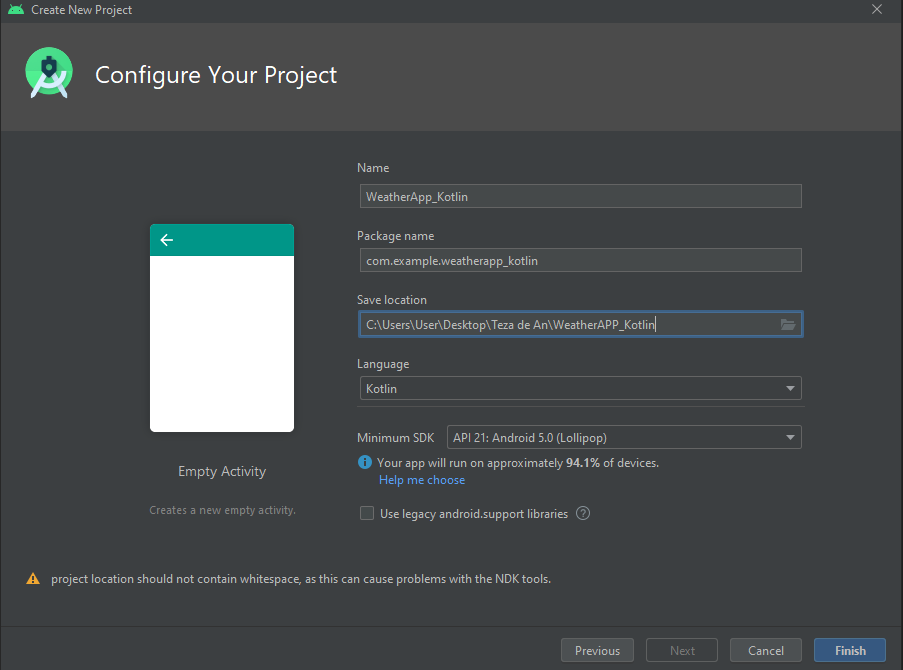


Fig. 3.1. Setările proiectului.

Pentru obținerea unui background cât de cât frumos, este creat un fișier resursă în directoriul res/drawable, ce are următorul conținut:

*<?xml version =”1.0” encoding=”utf-8”?>*

*<selector xmlns:android=”http://schemas.android.com/apk/res/android”>*

*<item>*

*<shape>*

*<gradient android:angle=”90” android:endColor=”#661323” android:startColor=”#1f1551”android:type=”linear” />*

*</shape>*

*</item>*

*</selector>*

Pentru a scăpa de meniul de sus ce strică design-ul aplicației, este folosită metoda setFlags, ce primește 2 parametri, (WindowManager.LayoutParams.FLAG\_FULLSCREEN este repetat de 2 ori, echivalentul său este o singură valoare, și anume setFlags(WindowManager...) deși fiind operații pe biți, al doilea parametru este oferit ca mască pentru operarea asupra câmpurilor în același timp, astfel se schimbă doar biții din mască, nu și din primu, acest procedeu este pentru obținerea eficienței de spațiu, dar și a performanței. Putea fi folosită în loc de virgulă operatorul | , dar este reduntant.

*supportActionBar?.hide()*

*window.setFlags( WindowManager.LayoutParams.FLAG\_FULLSCREEN,*

*WindowManager.LayoutParams.FLAG\_FULLSCREEN)*

*setContentView(R.layout.activity\_main)*

Următorul pas este crearea interfeței ce va afișa datele, este necesar de a avea 3 stări, și anume când se încarcă request-ul, atunci când request-ul nu lucrează sau apare vreo oarecare altă problemă. Astfel, se creează 2 layout-uri, în unul este interfața propriu zisă, iar în altul sunt cele 2 stări, care vor fi operate după starea din aplicație. Imaginile sunt atașate mai jos. De asemenea, este necesară crearea unui fișier uc resurse în res/values pentru string-urile din layout, și anume pentru evitarea warning-urilor, dar și pentru formatarea cifrelor obținute de pe server, astfel fișierul strings.xml conține:

*<resources>*

*<string name=”app\_name”>WeatherApp\_Kotlin</string>*

*<string name=”max\_temperature”>Max temperature</string>*

*<string name=”value”>Value</string>*

*<string name=”refresh”>Refresh</string>*

*<string name=”can\_t”>Can\’t</string>*

*<string name=”min\_temperature”>Min temperature</string>*

*<string name=”wind\_speed”>Wind Speed </string>*

*<string name=”updated”>Updated</string>*

*<string name=”tempValue”> %1$.2f ° C</string>*

*<string name=”updatedAt”>Updated at %1$d:%2$d</string>*

*</resources>*

Atenția este acordată string-ului cu numele tempValue și updatedAt, %1 înseamnă parametrul care va fi afișat, și $.2f este pentru convertirea parametrului dat în tipul de date double cu 2 puncte după virgulă, iar în updatedAt este primul parametru și al doilea descris ca număr întreg ($d).

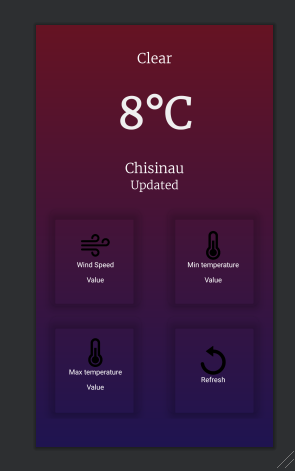


Fig. 3.2. Interfața aplicației.

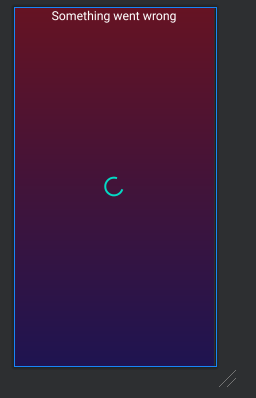


Fig. 3.3. Interfața la apariția problemelor.

Odată cu crearea layout-ului(fig. 3.2, 3.3), este necesar alegerea unui API, unul dintre cel mai popular API este *openweathermap* (fig 3.4)

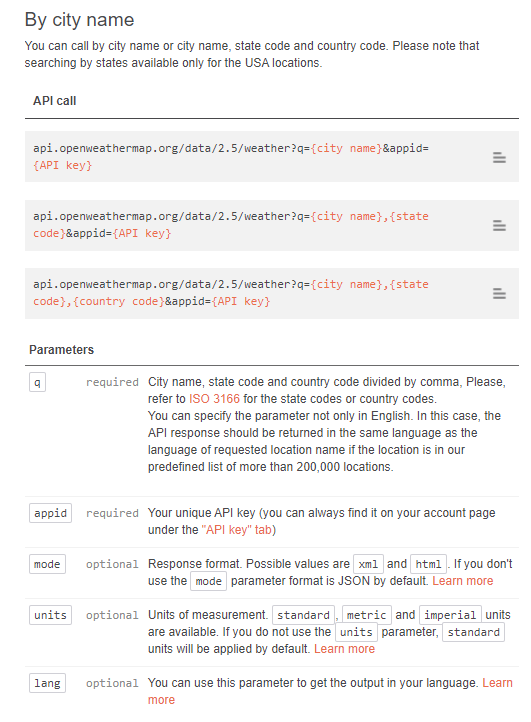


Fig. 3.4. Parametrii necesari pentru obținerea datelor.

Units reprezintă formatul de date care va fi primit de la server, este ales metric pentru obținerea datelor în grade Celsius. Însuși apelul la server are loc prin transmiterea parametrului city name, în cazul dat este Bălti, și cheia api.

Odată cu înțelegerea faptului cum are loc request-ul, este necesar de a adăuga compileOptions pentru compatibilitatea cu versiunea Java 1.8, dar și target-ul de asemenea, în build.gradle(Module: app) se adaugă:

*compileOptions {*

*sourceCompatibility JavaVersion.VERSION\_1\_8*

*targetCompatibility JavaVersion.VERSION\_1\_8*

*}*

*kotlinOptions {*

*jvmTarget = ”1.8**”*

*}*

Pentru crearea request-urilor este folosit Retrofit2, și coroutines din Kotlin, fiind o librărie foarte bună pe Kotlin în comparație cu cele existente pe Java, se folosesc pentru management-ul request-urilor asincrone, spre deosebire de java unde este blocat apelul, coroutines suspendă, dar și este mai comod de a fi folosit, și face codul mai citabil. Astfel, sunt adăugate la dependențe în build.gradle (Module:app).

*implementation ”org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.7”*

*implementation ”org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.5”*

*implementation ”com.squareup.retrofit2:retrofit:2.7.1”*

*implementation ”com.squareup.retrofit2:converter-gson:2.0.0”*

Aplicația necesită permisiune la internet pentru efectuarea adresărilor la server, astfel în AndroidManifest este necesar adăugarea acestei permisiuni.

<uses-permission android:name=”android.permission.INTERNET”/>

Este creat pachetul api pentru modelul de date ce vor fi primite de la adresare, utilizând plugin-ul Kotlin data class File from JSON, vor fi automat create în baza JSON-ului primit de pe server(fig. 3.6):



Fig. 3.5. Date obținute de pe API.

Plugin-ul creează mai multe clase, cu codul următor:

*WeatherJSON.kt*

*data class WeatherJSON(*

*val base: String,*

*val clouds: Clouds,*

*val coord: Coord,*

*val dt: Int,*

*val id: Int,*

*val main: Main,*

*val sys: Sys,*

*val timezone: Int,*

*val visibility: Int,*

*val weather: List<Weather>,*

*val wind: Wind*

*)*

*Clouds.kt*

*data class Clouds ( val all: Int)*

*Coord.kt*

*data class Coord( val lat: Double, val lon: Double)*

*Main.kt*

*data class Main (*

*@SerializedName(“feels\_like”)*

*Val feelsLike: Double,*

*val humidity: Int,*

*val pressure: Int,*

*val temp: Double,*

*@SerializedName(“temp\_max”)*

*val tempMax: Double,*

*@SerializedName(“temp\_min”)*

*val tempMin: Double*

*)*

*Sys.kt*

*data class Sys (*

*val country: String,*

*val id: Int,*

*val sunrise: Int,*

*val sunset: Int,*

*val type: Int*

*)*

*Weather.kt*

*Data class Weather(*

*Val description:String,*

*Val icon: String,*

*Val id: Int,*

*Val main: String*

*)*

*Wind.kt*

*Data class Wind(*

*Val deg: Int,*

*Val gust: Double,*

*Val speed: Double*

*)*

Retrofit-ul folosește aceste clase pentru primirea rezultatului adresării la server. Următorul pas este crearea unei interfețe pentru operarea cu retrofit.

*Interface APIRequests {*

*@GET(“weather?q=Balti&units=metric&appid=0dbb8c4b976379671419e66c70c93ded”)*

*fun getWeather(): Call<WeatherJSON>*

*}*

Este implementat retrofit2 Call, și metoda GET, pentru obținerea datelor de pe API estene necesară folosirea metodei GET, și conține orașul și cheia API. Urmat de funcție ce returnează apel la clasa de date creată recent.

Este creată funcția getCurrentData care va opera cu View-ul, dar și cu adresarea la server, de la început este setată starea de loading, adică este arătată pe display, fiind modificată vizibilitatea ei, și ascunsă vizibilitatea datelor, pentru a obține o animație de loading apoi de apariție a datelor.

*private fun getCurrentData() {*

*loaderView.isVisible = true*

*mainWindow.isVisible = false*

*}*

Următorul pas este crearea variabilei ce va crea adresarea Retrofit-ului, și anume

*val api = Retrofit.Builder().baseUrl(BASE\_URL)*

*.addConverterFactory(GsonConverterFactory.create())*

*.build().create(APIRequests::class.java)*

La apelul Retrofit.Builder este necesar de a transmite link-ul de bază, care este declarat ca o constantă variabilă

*const val BASE\_URL = “https:://api.openweathermap.org/data/2.5/”*

Este aplicată convertirea Gson, pentru citirea rezultatului transmiterea lui în clasele de date generate de plugin, apoi este aplicat build cu create la funcția din interfața creată, de asemenea se aplică ::class.java din cauza faptului că limbajul de programare Kotlin este proiectat de a putea lucra cu cod Java, aici este prezent Reflection.

După îndeplinirea acestor lucruri, este necesar utilizarea *Kotlin coroutines* pentru codul asincron fără de a fi blocat, dar și pornirea unui thread *GlobalScope.launch* care este responsabil de lucrul cu datele (Dispatchers.IO), construcția *withContext(Dispatchers.Main)* realizează aplicarea coroutinelor pentru codul asincron. Dacă răspunsul primit din partea serverului este unul bun *if(response.isSuccessful)*, are loc atribuirea variabilei *data* conținutul răspunsului de la server. Odată cu atribuirea curentă, se folosește coroutines *withContext(Dispatchers.Main*, care are ca scop operarea View-ului, și anume plasarea datelor în layout-ul creat *cityView.text = data.name...*

*GlobalScope.launch(Dispatchers.IO) {*

*try {*

*val response = api.getWeather().awaitResponse()*

*withContext(Dispatchers.Main) {*

*mainWindow.isVisible = true*

*loaderView.isVisible = false*

*cityView.text = data.name*

*windSpeedView.text = data.wind.speed.toString()*

*minTemperatureView.text =getString(R.string.tempValue ,data.main.tempMin)*

*weatherValueView.text = getString(R.string.tempValue, data.main.temp)*

*maxTemperatureView.text = getString(R.string.tempValue, data.main.tempMax)*

*weatherStatus.text = data.weather[0].main*

*updatedView.text = getString(*

*R.string.updatedAt,*

*Calendar.getInstance().get(Calendar.HOR\_OF\_DAY),*

*Calendar.getInstance().get(Calendar.MINUTE)*

*)*

*}*

*}*

*} catch (e: Exception) {*

*withContext(Dispatchers.Main) {*

*Toast.makeText(applicationContext, “Something wrong”, Toast.LENGTH\_SHORT*

*).show()*

*mainWindow.isVisible = false*

*errorText.isVisible = true*

*}*

*}*

Are loc aplicarea construcției try-catch, în ambele cazuri este folosirea *Kotlin coroutines* pentru management-ul aplicației, și anume stările, dar și apariția pe ecran a datelor, astfel dacă răspunsul este unul greșit sau lipsește conexiunea de internet, aplicația trece la un regim de eroare(fig. 3.4). Codul sursă a aplicației elaborate pe Kotlin este disponibil on-line pe adresa: https://github.com/VitalieVS/WeatherApp\_Kotlin

* 1. Elaborarea aplicației în limbajul de programare Java

Fișierele de resurse strings.xml, imaginile, background-ul și interfața se implementează în aplicația pe Java și se obține aceeași interfață (fig. 3.7, fig. 3.4). Pentru Java este folosită o clasă interioară, care moștenește de la clasa AsyncTask și care va rescrie metodele *onPreExecute*, *doInBackground* și *onPostExecute.* Diferența dintre aplicația pe Kotlin este lucrul cu interfața, și anume sunt necesare de a declara toate câmpurile din ea

*@SuppressLint("StaticFieldLeak")*

*private static class WeatherTask extends AsyncTask<String, Void, String> {*

*ProgressBar progressBar;*

*LinearLayout mainWindow;*

*CardView refreshCardView;*

*TextView errorText;*

*TextView cityView;*

*TextView windSpeedView;*

*TextView minTemperatureView;*

*TextView weatherValueView;*

*TextView maxTemperatureView;*

*TextView weatherStatusView;*

*TextView updatedAtView;*

*MainActivity currentActivity;*

De asemenea există necesitatea de a declara o variabilă String sub numele de oraș în cazul când este dorință de a schimba orașul pentru care se vor primi datele, dar și este necesar adăugarea unei referințe la clasa principală, transmiterea acestor două are loc prin constructor.

*private String CITY;*

*private WeakReference<MainActivity> activityReference;*

*WeatherTask(MainActivity context, String city) {*

*this.CITY = city;*

*this.activityReference = new WeakReference<>(context);*

*}*

Urmează rescrierea metodei onPreExecute care va obține referința curentă și cu referința va căuta elementele pentru ascunderea stării aplicației de erori (fig. 3.4) și afișarea (fig. 3.3) cu datele necesare.

*@Override*

*protected void onPreExecute() {*

*super.onPreExecute();*

*currentActivity = activityReference.get();*

*if (currentActivity == null || currentActivity.isFinishing()) return;*

*mainWindow = currentActivity.findViewById(R.id.mainWindow);*

*progressBar = currentActivity.findViewById(R.id.loaderView);*

*errorText = currentActivity.findViewById(R.id.errorText);*

*mainWindow.setVisibility(View.GONE);*

*progressBar.setVisibility(View.VISIBLE);*

*errorText.setVisibility(View.GONE);*

*}*

Este necesară rescrierea metodei *doInBackground* pentru apelarea către server și obținerea răspunsului

*@Override*

*protected String doInBackground(String... strings) {*

*StringBuilder content = new StringBuilder();*

*try {*

*URL url = new URL(*

*BASE\_URL +*

*"weather?q=" +*

*this.CITY +*

*"&units=metric&appid=0dbb8c4b976379671419e66c70c93ded");*

*URLConnection connection = url.openConnection();*

*BufferedReader reader = new BufferedReader(newInputStreamReader(connection.getInputStream()));*

*String line;*

*while ((line = reader.readLine()) !=null)content.append(line).append("\n");*

*} catch (IOException e) {*

*return e.toString();*

*}*

*return content.toString();*

*}*

Iar după executarea metodei este necesar de rescriere a metodei *onPostExecute* care se apelează după finisarea metodei de mai sus, folosită pentru prelucrarea și afișarea răspunsului primit de la API

*@RequiresApi(api = Build.VERSION\_CODES.O)*

*@Override*

*protected void onPostExecute(String s) {*

*super.onPostExecute(s);*

*try {*

*final JSONObject jsonObj = new JSONObject(s);*

*final JSONObject main = jsonObj.getJSONObject("main");*

*final JSONObject sys = jsonObj.getJSONObject("sys");*

*final JSONObject wind = jsonObj.getJSONObject("wind");*

*final JSONObject weather = jsonObj.getJSONArray("weather")*

*.getJSONObject(0);*

*final String updatedAtText = "Updated At:" +*

*LocalTime.now().getHour() + ":" + LocalTime.now().getMinute();*

*final String temp = main.getString("temp") + "°C";*

*final String tempMin = "Min Temp: " + main.getString("temp\_min") + "°C";*

*final String tempMax = "Max Temp: " + main.getString("temp\_max") + "°C";*

*final String windSpeed = wind.getString("speed");*

*final String weatherDescription = weather.getString("description");*

*final String address = jsonObj.getString("name") + ", " +*

*sys.getString("country");*

*windSpeedView = currentActivity.findViewById(R.id.windSpeedView);*

*minTemperatureView =currentActivity.findViewById(R.id.minTemperatureView);*

*maxTemperatureView=currentActivity.findViewById(R.id.maxTemperatureView);*

*weatherStatusView = currentActivity.findViewById(R.id.weatherStatus);*

*updatedAtView = currentActivity.findViewById(R.id.updatedView);*

*cityView = currentActivity.findViewById(R.id.cityView);*

*weatherValueView = currentActivity.findViewById(R.id.weatherValueView);*

*refreshCardView = currentActivity.findViewById(R.id.refreshCardView);*

*cityView.setText(address);*

*windSpeedView.setText(windSpeed);*

*minTemperatureView.setText(tempMin);*

*weatherValueView.setText(temp);*

*maxTemperatureView.setText(tempMax);*

*weatherStatusView.setText(weatherDescription);*

*updatedAtView.setText(updatedAtText);*

*mainWindow.setVisibility(View.VISIBLE);*

*progressBar.setVisibility(View.GONE);*

*} catch (Exception e) {*

*progressBar.setVisibility(View.VISIBLE); errorText.setVisibility(View.VISIBLE);*

*}*

*}*

*}*

Se observă multiple linii de cod care pe Kotlin sunt ușor evitate și codul devine mai ușor de a fi citit, pe Java însă acest aspect lipsește. Atât codul cât și volumul pare mult mai complicat în Java decât pe Kotlin. Codul sursă a aplicației elaborate pe Java este disponibil on-line pe: https://github.com/VitalieVS/WeatherAPP\_Java

CONCLUZII

Elaborarea aplicației pe Kotlin a luat puțin timp, datorită limbajului, și a plusurilor sale, astfel aplicația lucrează corect, nu prezintă bug-uri și cu o performanță bună.

BIBLIOGRAFIE

1. Алина Уткина. Java vs Kotlin для Android-разработки: ответы за и против.

[on-line] [citat 12 oct. 2020]. Disponibil: <https://tproger.ru/articles/java-vs-kotlin/>

1. Вероника. Плюсы и минусы программирования на Java. [on-line] [citat 15 oct.2020]. Disponibil:<https://medium.com/nuances-of-programming/%D0%BF%D0%BB%D1%8E%D1%81%D1%8B-%D0%B8-%D0%BC%D0%B8%D0%BD%D1%83%D1%81%D1%8B-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D0%BD%D0%B0-java-2861f4c2a0d5>
2. Juhani Lehtimaki. Why starting a new Android project with Java is a bad idea. [on-line] [citat 16 oct. 2020]. Disponibil: <https://medium.com/snapp-mobile/why-starting-a-new-android-project-with-java-is-a-bad-idea-359bffe0bbd6>
3. Shubham Rattra. Kotlin VS Java – Android Development. [on-line] [citat 20 oct. 2020]. Disponibil: <https://hackernoon.com/kotlin-vs-java-android-development-qh6z329j>
4. Navdeep Singh Gill. Kotlin vs Java, which is better for Android Development?. [on-line] [citat 22 oct. 2020]. Disponibil: <https://www.xenonstack.com/blog/kotlin-andriod/>
5. Mikhail Raevskiy. Java vs Kotlin for Android Development – 16 Pros and Cons. [on-line] [citat 25 dec. 2020]. Disponibil: <https://medium.com/dev-genius/java-vs-kotlin-for-android-development-16-pros-and-cons-9e046f7a4908>
6. Hiral Atha. Java Vs Kotlin – Which should you choose for Android Development.[on-line] [citat 21 ian. 2021]. Disponibil: <https://www.moveoapps.com/blog/java-vs-kotlin/>
7. Avinash Sharma. Know why these 15 famous apps migrated from Java to Kotlin. [on-line] [citat 04 febr. 2021]. Disponibil: <https://appinventiv.com/blog/apps-migrated-from-java-to-kotlin/>