
Universitatea „Petrol – Gaze” din Ploieşti

Calculator decimal/hexadecimal

Realizat de: Coşereanu Emanuel, INFO an 2, grupa 40322

Cuprins

<i>Introducere</i>	3
<i>Design</i>	3
ActionBar pentru MainActivity cu butonul Istoric.....	3
ActionBar pentru HistoryActivity cu butoanele Email și Calcul.....	5
TextAnterior(TextA) și TextCurent(TextC) design	7
Cele 16 butoane	9
Butonul 10/16	9
Butonul BACK	13
Butoanele pentru operatorii +, -, *	13
Butonul „=” design.....	14
Istoricul (Ecranul 2).....	16
<i>Funcționalitate</i>	17
TextAnterior(TextA) și TextCurent(TextC) funcționalitate, butonul „=”	18
Operatorii +, - și *	19
Butonul „=” funcționalitate	20
Selectarea unui element din istoric	20
Colorarea și inserarea elementelor în istoric.....	22
Bara de scroll și scroll automat la ultimele valori introduse	23
Convertirea numerelor la schimbarea bazei.....	24
<i>Observații</i>	25
Modul Landscape	25
Resetarea calculatorului când apăs pe 0-9, A-F; limita numerelor	25
Modificare operatorilor	26

Introducere

Aplicația dezvoltată se numește „Calculator_decimal_hexadecimal” și a fost proiectată pentru a permite utilizatorului să realizeze calcule atât în baza 10, cât și în baza 16.

Am încercat respectarea cu cea mai mare strictețe posibilă a cerințelor, însă am realizat și câteva mici modificări de design sau de funcționalitate pentru a oferi o notă de originalitate sau de practicabilitate. Toate modificările vor fi prezentate în cuprinsul documentației, dar mai ales în secțiunea de **Observații**.

Documentația va fi împărțită în două mari secțiuni: *Design* și *Funcționalitate* pentru a fi mult mai ușor de urmărit respectarea (sau nu) a tuturor cerințelor.

Design

În ceea ce privește design-ul, modificările pe care le-am făcut nu sunt foarte mari. Am încercat să dau o notă de originalitate aplicației prin culoarea verde închis a butoanelor și a ActionBar-ului, în rest am păstrat culorile standard din cerințe pentru butoane.

ActionBar pentru MainActivity cu butonul Istoric

Pentru a crea bara de meniu, am creat folderul **menu** în **res**, urmând ca în acesta să creez fișierul **menu_main.xml** unde am scris codul pentru bara de meniu și butonul **Istoric**(Fig. 1.1). Pentru ca bara de meniu să fie vizibilă am modificat tema din **res/values/themes/themes.xml** din **Theme.Material3.DayNight.NoActionBar** în **Theme.Material3.DayNight**, iar în **MainActivity.kt** am încărcat meniul și am gestionat acțiunile (adică când este apăsat butonul **Istoric** să fim trimiși către al doilea ecran (**HistoryActivity.kt**))(Fig. 1.2).

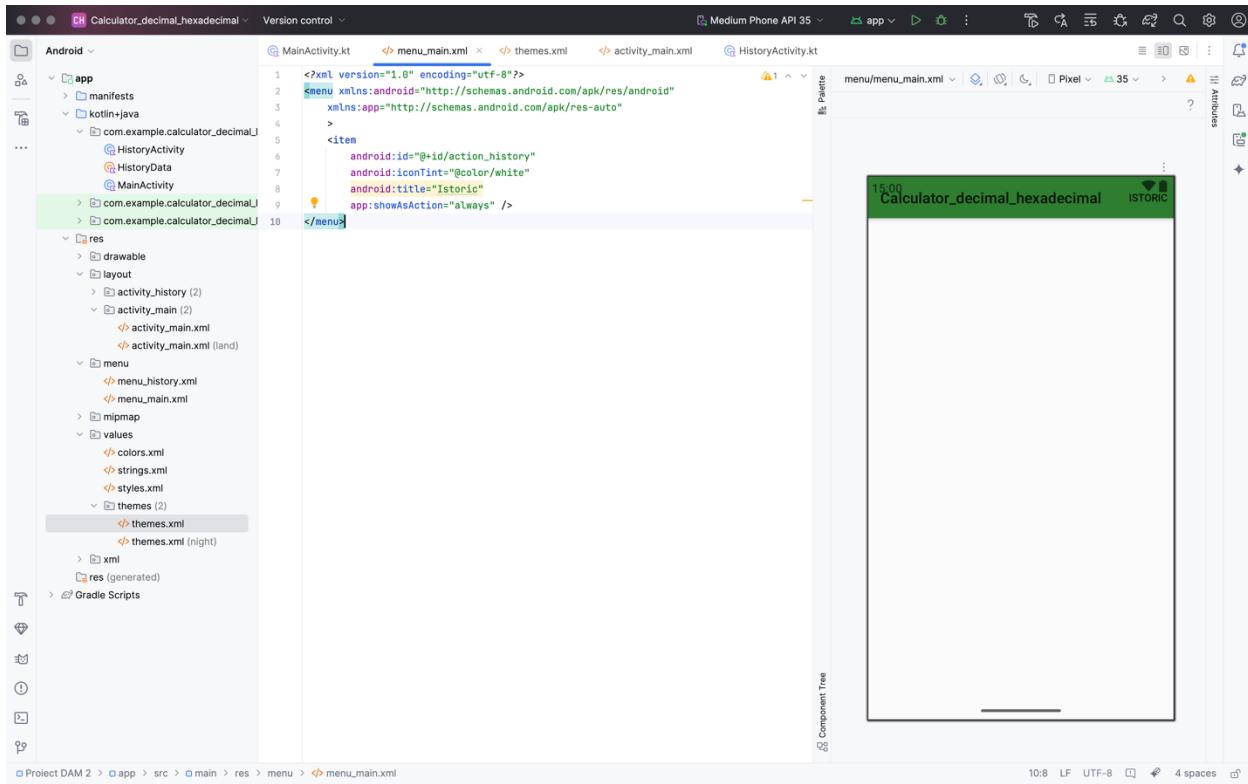


Fig. 1.1 Bara de meniu pentru MainActivity.kt

```
//Incarcam meniul
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
    return true
}

//Gestionam actiunile din meniu
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId){
        R.id.action_history -> {
            //Cand apasam pe "Istoric", deschidem HistoryActivity
            val intent = Intent(packageContext: this, HistoryActivity::class.java)
            historyLauncher.launch(intent)
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Fig. 1.2 Încărcarea meniului în ecranul 1 și setarea acțiunilor

ActionBar pentru HistoryActivity cu butoanele Email și Calcul

Pentru a crea bara de meniu pentru ecranul 2, am urmat aceeași pașă ca mai sus, doar că am creat fișierul **menu_history.xml** unde am scris codul pentru butoane și bară(Fig. 1.3). Am încărcat meniul în HistoryActivity.kt setând acțiunile necesare pentru fiecare buton(Fig. 1.4): Calcul – ne duce către ecranul 1, Email – ne permite să trimitem istoricul pe mail(Fig. 1.5) (Am creat o funcție pentru asta (Fig. 1.6)).

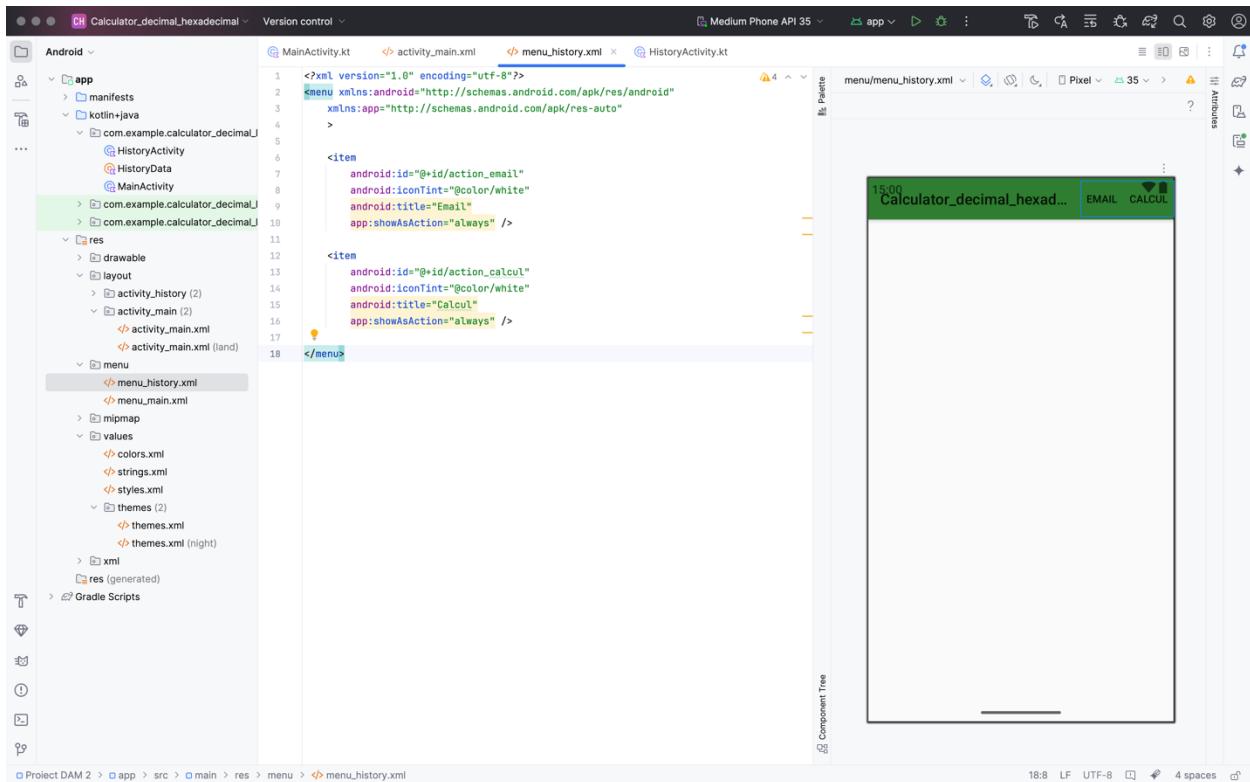


Fig. 1.3 Bara de meniu pentru HistoryActivity.kt

```

//Gestionam acțiunile din meniu
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        //Acest id corespunde cu butonul "calcul"
        R.id.action_calcul -> {
            //Inchidem HistoryActivity si deschidem activitatea principală (MainActivity)
            finish()
            true
        }
        R.id.action_email -> {
            //Se trimite emailul cu istoricul la apasarea butonului email
            sendEmail()
            return true
        }
        else -> super.onOptionsItemSelected(item)
    }
}

```

Fig. 1.4 Încărcarea meniului în ecranul 2 și setarea acțiunilor

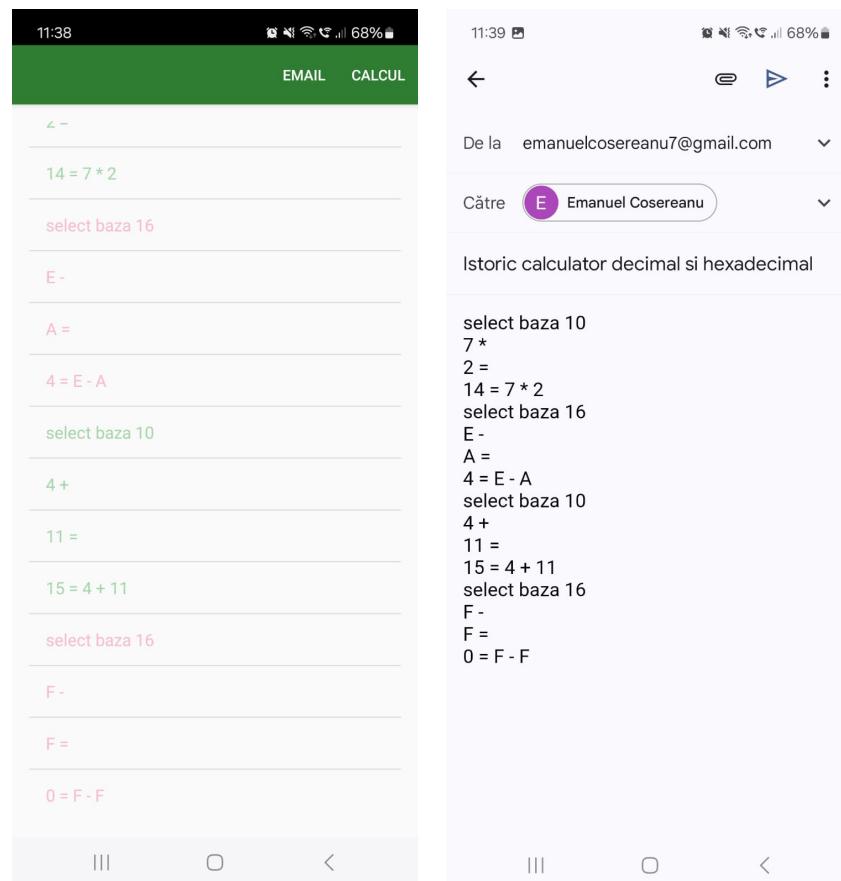


Fig. 1.5 Trimiterea istoricului pe email

```

private fun sendEmail() {

    //obtinem continutul istoricului, dupa care punem fiecare element pe o linie separata pentru a putea sa l trimitem pe mail
    val HistoryList = HistoryData.getAll()
    val emaiBody = HistoryList.joinToString( separator: "\n") { it.toString() }

    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "message/rfc822" //spune androidului ca vom trimite un mesaj de tip email rfc822 = Standard Format for ARPA Internet Text Messages
    intent.putExtra(Intent.EXTRA_EMAIL, arrayOf("emanuelcosoreanu7@gmail.com"))
    intent.putExtra(Intent.EXTRA_SUBJECT, value: "Istoric calculator decimal si hexadecimal") //titlul emailului
    intent.putExtra(Intent.EXTRA_TEXT, emaiBody) //corpul emailului

    //daca putem trimite emailul o face, daca nu,afisam ca am intampinat erori (nu s-au gasit aplicatii sa trimitem mailul)
    try {
        startActivity(Intent.createChooser(intent, title: "Trimteti emailul..."))
    } catch (e: android.content.ActivityNotFoundException) {
        Toast.makeText(context: this, text: "Nu exista aplicatii pentru trimiterea emailului", Toast.LENGTH_SHORT).show()
    }
}

```

Fig. 1.6 Funcție pentru trimiterea istoricului pe mail

TextAnterior(TextA) și TextCurent(TextC) design

Am creat 2 TextView-uri aproape identice aşa cum se poate vedea în codul pus mai jos doar că TextA are inițial valoarea 0 afișată. Primul TextView este folosit pentru a „memora” numerele după ce s-a apăsat unul dintre operatori, iar al doilea este folosit pentru a vedea numerele introduse în momentul curent. Culoarea din TextView-uri se va modifica în funcție de baza în care se aflăm. Aceste funcționalități vor fi prezentate în Fig. 1.7 și Fig. 1.8.

```

<TextView
    android:id="@+id/textAnterior"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:backgroundTint="@color/white"
    android:textAlignment="viewEnd"
    android:text="0"
    android:textColor="@color/pastel_green"
    android:maxLines="1"
    android:textSize="34sp" />

<TextView
    android:id="@+id/textCurent"
    android:layout_width="match_parent"

```

```

    android:layout_height="100dp"
    android:backgroundTint="@color/white"
    android:textAlignment="viewEnd"
    android:textColor="@color/pastel_green"
    android:maxLines="1"
    android:textSize="34sp" />

```

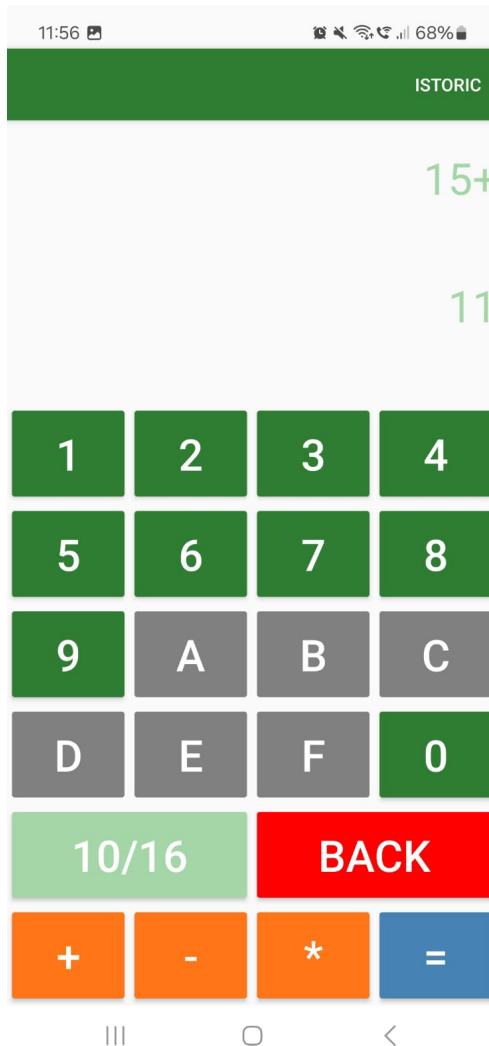


Fig. 1.7 Culoarea textului din TextAnterior și TextCurrent când suntem în baza 10

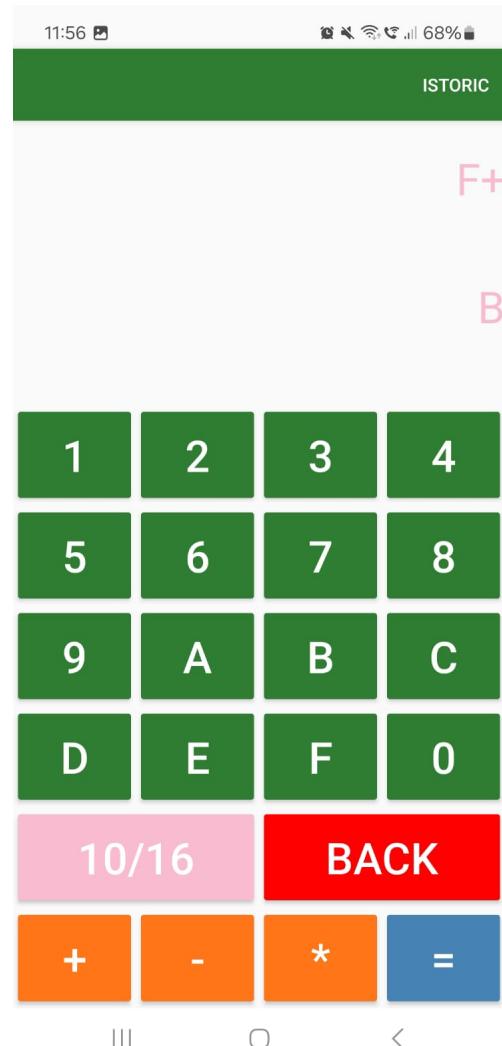


Fig. 1.8 Culoarea textului din TextAnterior și TextCurrent când suntem în baza 16

Cele 16 butoane

Butoanele au fost colorate verde pentru a da o notă de originalitate. Butoanele de la A la F sunt inițial gri și dezactivate pentru că ne aflăm în baza 10, așa cum se poate observa din codul pus mai jos. Despre cum le-am creat, așezat și setat culoarea nu voi mai intra în detalii, deoarece este la fel ca la tema trecută.

```
//dezactivam butoanele de la A la F pentru ca initial aplicatia  
este in baza 10  
buttonA.isEnabled = false  
buttonB.isEnabled = false  
buttonC.isEnabled = false  
buttonD.isEnabled = false  
buttonE.isEnabled = false  
buttonF.isEnabled = false  
  
//setam butoanele cu gri atunci cand suntem in baza 10 pentru ca  
initial suntem in baza 10  
buttonA.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)  
buttonB.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)  
buttonC.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)  
buttonD.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)  
buttonE.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)  
buttonF.setBackgroundTintList =  
resources.getColorStateList(R.color.gray, theme)
```

Butonul 10/16

La apăsarea acestui buton, deoarece suntem inițial în baza 10, vom activa baza 16, deci butonul va deveni roz, vom activa și colora în verde butoanele de la A la F și vom converti în baza 16 numerele care se află în TextAnterior și TextCurrent, dacă se află.

```

button10_16.setOnClickListener {
    isHexMode = !isHexMode

    //activam sau dezactivam butoanele de la A la F
    buttonA.isEnabled = isHexMode
    buttonB.isEnabled = isHexMode
    buttonC.isEnabled = isHexMode
    buttonD.isEnabled = isHexMode
    buttonE.isEnabled = isHexMode
    buttonF.isEnabled = isHexMode

    //schimbam culoarea butonului si textului + convertim
    numerele + schimbam limita textului pe care il putem adauga in
    textCurrent
    if(isHexMode) {

        //limita in baza 16
        maxLength = 12

        //setam butoanele cu verde atunci cand suntem in baza 16
        buttonA.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)
        buttonB.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)
        buttonC.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)
        buttonD.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)
        buttonE.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)
        buttonF.setBackgroundTintList =
            resources.getColorStateList(R.color.green, theme)

        button10_16.setBackgroundTintList =
            resources.getColorStateList(R.color.pastel_pink, theme)

textAnterior.setTextColor(resources.getColor(R.color.pastel_pink))
}

textCurrent.setTextColor(resources.getColor(R.color.pastel_pink))

```

```

//convertim textul din hexadecimal in decimal
val currentText = textCurent.text.toString()
val anteriorText = textAnterior.text.toString()

//vedem daca in textAnterior avem un rezultat, adica
daca avem semnul =, pentru ca atunci cand convertim sa il
adaugam
    var e_egal = ""
    if(textAnterior.text.first() == '=')
        e_egal = "="

//convertim textul curent si textul anterior in noua
baza
    val newTextCurent = convertToBase(currentText, 10, 16)
    val newTextAnterior = convertToBase(anteriorText, 10,
16)

//Actualizam textul
textCurent.text = newTextCurent
if(e_egal == "=") {
    textAnterior.text = "= $newTextAnterior"
}
else {
    textAnterior.text = newTextAnterior
}
}
else {

//limita in baza 10
maxLenght = 15

//setam butoanele cu gri atunci cand suntem in baza 10
buttonA.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)
buttonB.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)
buttonC.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)
buttonD.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)

```

```

buttonE.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)
buttonF.backgroundTintList =
resources.getColorStateList(R.color.gray, theme)

button10_16.backgroundTintList =
resources.getColorStateList(R.color.pastel_green, theme)

textAnterior.setTextColor(resources.getColor(R.color.pastel_gree
n))

textCurent.setTextColor(resources.getColor(R.color.pastel_green)
)

//convertim textul din hexadecimal in decimal
val currentText = textCurent.text.toString()
val anteriorText = textAnterior.text.toString()

//convertim textul curent si textul anterior in noua
baza
val newTextCurent = convertToBase(currentText, 16, 10)
val newTextAnterior = convertToBase(anteriorText, 16,
10)

//vedem daca in textAnterior avem un rezultat, adica
daca avem semnul =, pentru ca atunci cand convertim sa il
adaugam
var e_egal = ""
if(textAnterior.text.first() == '=')
    e_egal = "="

//Actualizam textul
textCurent.text = newTextCurent
if(e_egal == "=") {
    textAnterior.text = "= $newTextAnterior"
}
else {
    textAnterior.text = newTextAnterior
}

```

```
    }  
}
```

Butonul BACK

Butonul BACK este asemănător cu cel din tema trecută, având aceeși funcționalitate, șterge ultimul caracter introdus în TextCurent, doar că dacă nu avem niciun caracter în TextC, ne pune automat un 0.

```
this.buttonBack.setOnClickListener {  
    val currentText = textCurent.text.toString()  
    if(currentText.isNotEmpty())  
    {  
        textCurent.text = currentText.substring(0,  
currentText.length - 1)  
        // creeam un substring din stringul principal minus  
        ultimul caracter  
        if(textCurent.text.toString().isEmpty())  
            textCurent.text = "0"  
    }  
    else  
        textCurent.text = "0"  
}
```

Butoanele pentru operatorii +, -, *

Butoanele se află pe ultimul rând din Table Layout, alături de butonul „=” și la apăsarea lor vom trimite textul din TextC în TextA dacă nu se află încă ceva în TextA și setăm TextC cu valoarea 0. În cazul în care avem deja un număr în TextA verificăm dacă acesta este un rezultat sau un operand și actualizăm după caz cu operatorul pe care l-am selectat. Voi lasă mai jos doar codul pentru butonul „+”, celelalte butoane fiind asemănătoare.

```

buttonPlus.setOnClickListener { v: View? ->
    val anteriorText = textAnterior.text.toString()
    val currentText = textCurent.text.toString()

    if(anteriorText.first() == '=') {
        //daca avem egal (adica daca avem un rezultat si dorim sa
        continuam sa calculam cu el, adaugam operatorul
        textAnterior.text = (anteriorText.substring(2,
        anteriorText.length))
        textAnterior.append("+")
    }
    else if(anteriorText.last() == '-' || anteriorText.last() ==
    '+' || anteriorText.last() == '*') {
        //daca dorim sa schibam operatorul putem face asta aici
        //ori daca apasam din nou din greseala pe acelasi
        operator, acesta ramane neschimbat
        textAnterior.text = (anteriorText.substring(0,
        anteriorText.length - 1))
        textAnterior.append("+")
    }
    else if(currentText.isNotEmpty()) {
        textAnterior.text = "$currentText"
        textCurent.text = "0"
    }
    //activam butonul =
    buttonEqual.isEnabled = true
}

```

Butonul „=” design

Design-ul la acest buton este același ca la cele de la 0 la 9 și de la A la F, diferența dintre ele fiind doar culoarea, butonul acesta fiind albastru. Pentru a respecta cerințele de funcționalitate butonul este dezactivat cât timp nu este nimic în TextAnterior și va deveni activ în momentul în care putem efectua un calcul.

La apăsarea butonului vom prelua ce avem scris în TextA și vom salva în 2 variabile numărul și operatorul, respectiv vom prelua numărul scris în TextC, iar în funcție de operatorul prezent vom efectua calculul și rezultatul va fi afișat în TextA.

Tot la apăsarea butonului „=” am decis să trimit ambii operanzi către istoric, rezultatul și baza în care s-a realizat calculul pentru a nu supraîncărca istoricul de fiecare dată când schimb baza sau operatorul. Voi lăsa codul pentru acest buton mai jos.

```
buttonEqual.setOnClickListener {
    val operator = textAnterior.text.toString().last()
    val number1 = textAnterior.text.toString().substring(0,
textAnterior.text.toString().length - 1)
    val number2 = textCurrent.text.toString()

    val base = if(isHexMode) 16 else 10 //daca suntem in baza 16
base va fi 16 altfel 10

    val num1 = BigInteger(number1, base) //convertim number1
care era un string la baza de care avem nevoie
    val num2 = BigInteger(number2, base)

    val result: BigInteger = when (operator) {
        '+' -> num1 + num2
        '-' -> num1 - num2
        '*' -> num1 * num2
        else -> BigInteger.ZERO
    }

    //rezultatul va fi convertit inapoi in baza actuala
    val resultText = if(isHexMode)
Integer.toHexString(result.toInt()).uppercase() else
result.toString()

    textAnterior.text = "= $resultText"
    textCurrent.text = "0"
    buttonEqual.isEnabled = false

    //vedem baza in care suntem pentru a stii ce afisam
    val baseText = if(isHexMode) "select baza 16" else "select
baza 10"

    //adaugam in istoric operatia si numerele
    HistoryData.addOperation(number1, operator.toString(),
```

```

        number2, resultText, baseText, this)
    }
}

```

Istoricul (Ecranul 2)

Pentru a avea istoricul pe un ecran separat am creat altă activitate denumită **HistoryActivity.kt** cu fișierul XML reprezentativ **activity_history.xml** unde am creat un LinearLayout și un ListView în interiorul acestuia(Fig. 1.9).

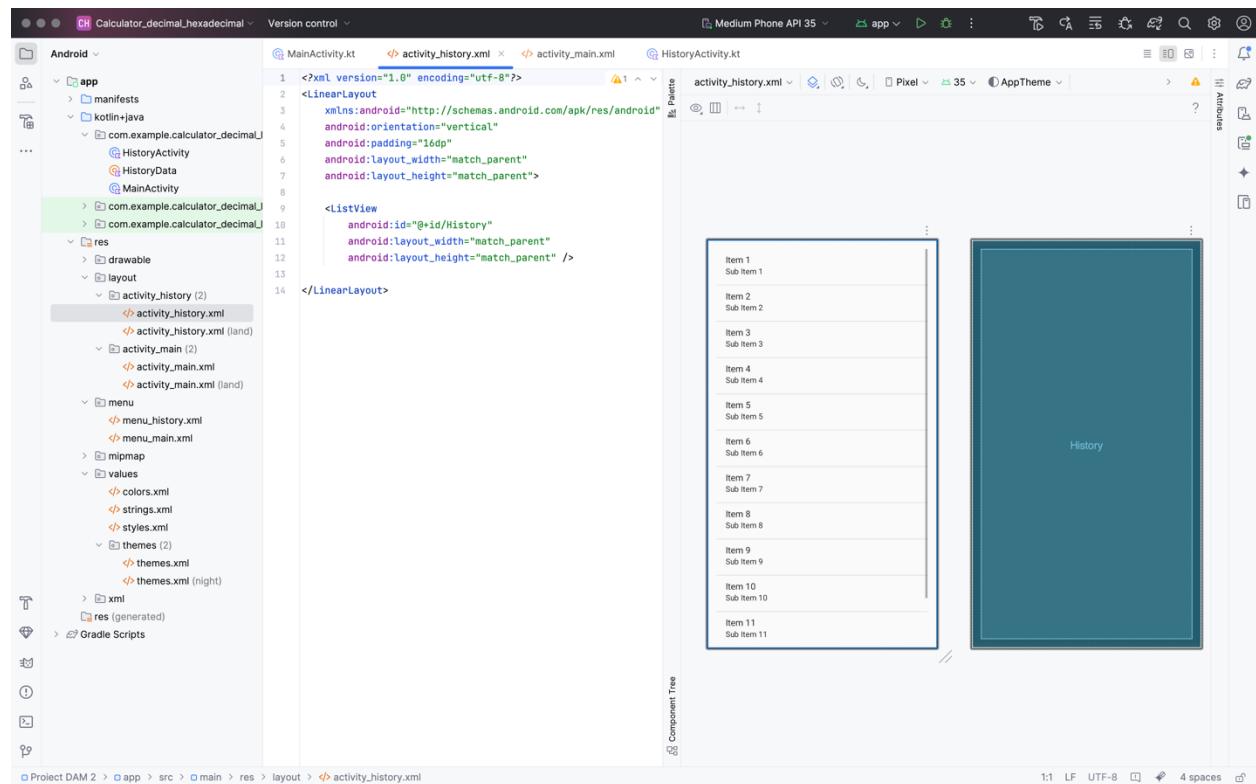


Fig. 1.9 Crearea istoricului

Funcționalitate

Pentru că unele aspecte ce reprezintă funcționalitatea aplicației au fost prezentate mai sus, voi încerca să prezint ce nu a fost spus, ori să completez aspectele descrise în partea Design.

Așa cum a fost prezentat mai sus, butoanele de la A la F sunt dezactivate inițial pentru că suntem în baza 10, și devin active și verzi când suntem în baza 16. Pentru a știi în ce bază ne aflăm vom folosi o variabilă denumită **isHexMode** care are valoarea *true* când suntem în baza 16 și *false* când suntem în baza 10.

`buttonA.isEnabled = isHexMode` – activează sau dezactivează butonul A în funcție de baza în care sunt.

`buttonA.backgroundTintList = resources.getColorStateList(R.color.green, theme)` – dacă suntem în baza 16 vom schimba culoarea butonului în verde, dacă suntem în baza 10 culoarea va fi gri. Culoarele au fost declarate în fișierul **res/values/colors.xml** (Fig. 2.1).

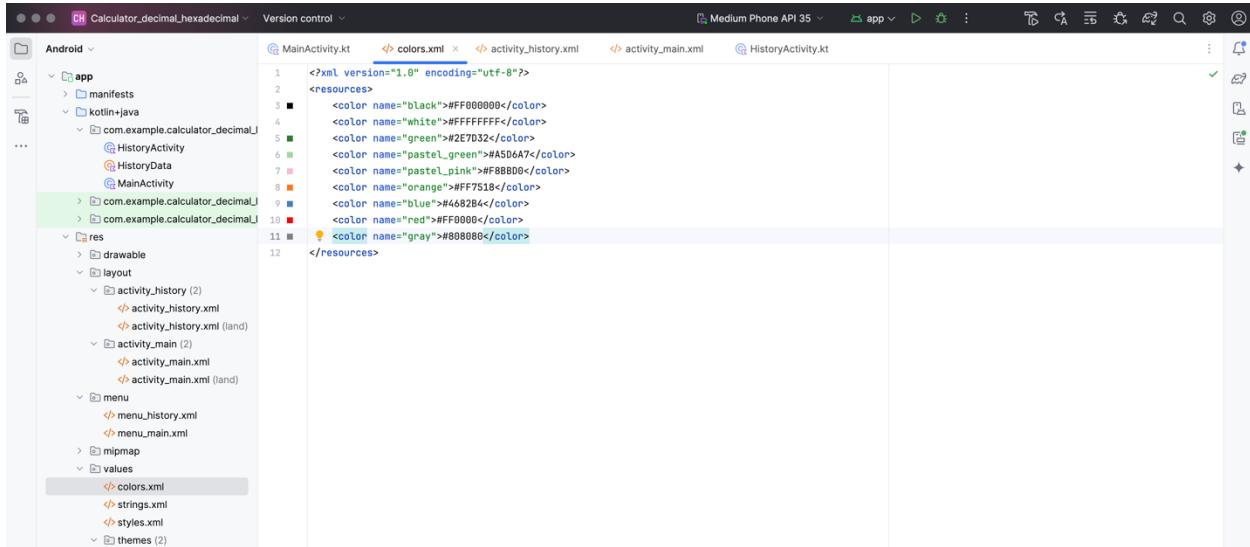


Fig. 2.1 Declararea culorilor

TextAnterior(TextA) și TextCurent(TextC) funcționalitate, butonul „=”

Pentru a avea inițial valoarea 0 în TextA am modificat codul XML în felul următor:

```
android:text="0"
```

În TextC nu am pus inițial niciun text.

Imediat după declararea și inițializarea butonului Egal, l-am dezactivat, urmând să îl activez doar la apăsarea unui buton dintre: -, + sau *, schimbând atributul *isEnabled* în **true**.

```
buttonEqual.isEnabled = false
```

Pentru a nu permite să avem numere de forma 007 sau 00EFF (adică care încep cu 0) în TextC și respectiv în TextA am folosit următorul artificiu de programare:

- Dacă doresc să adaug o cifră sau o literă în TextC, iar în interiorul acestuia avem valoarea 0, înlocuim acea valoarea cu tasta apăsată.
- Dacă doresc să adaug o cifră sau o literă în TextC și avem deja în el un număr diferit de 0, pur și simplu lipim caracterul tastat de cele deja prezente.

Voi prezenta această funcționalitate pentru butonul 9 mai jos.

```
button9.setOnClickListener { v: View? ->
    val currentText = textCurent.text.toString()
    if(textAnterior.text.last() != '-' &&
textAnterior.text.last() != '+' && textAnterior.text.last() != '*') //vreau ca atunci cand in textAnterior am un rezultat si
apas pe o tasta care nu e operator, sa resetez calculatorul
        textAnterior.text = "0"
    if (currentText == "0") {
        textCurent.text = "9"
    } else {
        if(textCurent.text.length < maxLenght)
            appendToTextView("9")
    }
}
```

Operatorii +, - și *

La apăsarea unui operator, dacă în TextA nu avem un număr introdus, trimitem numărul din TextC către TextA alături de operatorul apăsat. Dacă avem deja un număr și un operator în TextA schimbăm operatorul în funcție de tasta apăsată, iar dacă în TextA avem un rezultat(adică avem semnul „=” ca prim caracter) preluăm acel număr și îi asociem operatorul selectat pentru a continua calculul folosind numărul „rezultat”.

```
buttonMinus.setOnClickListener { v: View? ->
    val anteriorText = textAnterior.text.toString()
    val currentText = textCurrent.text.toString()

    if(anteriorText.first() == '=') {
        //daca avem egal (adică daca avem un rezultat si dorim sa
        continuam sa calculam cu el, adaugam operatorul
        textAnterior.text = (anteriorText.substring(2,
        anteriorText.length))
        textAnterior.append("-")
    }
    else if(anteriorText.last() == '-' || anteriorText.last() ==
    '+' || anteriorText.last() == '*') {
        //daca dorim sa schibam operatorul putem face asta aici
        //ori daca apasam din nou din greseala pe același
        operator, acesta ramane neschimbăt
        textAnterior.text = (anteriorText.substring(0,
        anteriorText.length - 1))
        textAnterior.append("-")
    }
    else if(currentText.isNotEmpty()) {
        textAnterior.text = "$currentText-"
        textCurrent.text = "0"
    }
    //activam butonul =
    buttonEqual.isEnabled = true
}
```

Din câte se poate vedea cu 2 rânduri mai sus, la apăsarea unui operator activăm și butonul „=”.

Butonul „=” funcționalitate

Codul pentru acest buton, cât și o parte din funcționalitatea lui a fost deja prezentată în partea de design, aşa că în această parte voi încerca să explic și să arăt de ce am modificat puțin cerința și de ce trimis totul către istoric doar când apăs butonul (Fig. 2.2, Fig. 2.3). Mi s-a părut că voi încărca prea tare istoricul dacă trimis mereu către acesta orice modificare pe care o fac, în plus, am ales să trimis mereu către istoric și baza în care s-a realizat calculul.

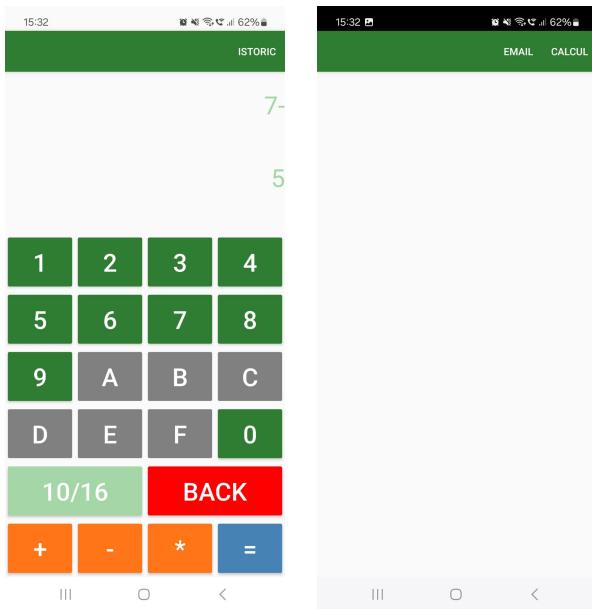


Fig. 2.2 Istoricul înainte de a apăsa butonul „=”

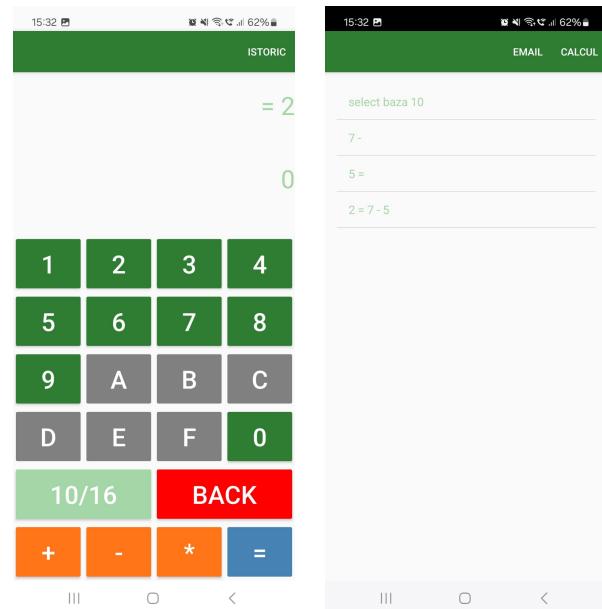


Fig. 2.3 Istoricul după apăsarea butonului „=”

Selectarea unui element din istoric

Pentru a selecta un element din istoricul am adăugat un **setOnItemClickListener** pentru ListView-ul ce reprezintă istoricul, memorând într-o variabilă linia selectată. Am presupus că pe orice linie ce conține numere am apăsat, vom alege mereu primul element (ex: $7 = 5 + 4$, vom alege numărul 7, (Fig. 2.4)). Pentru a afla baza în care se află numărul vom merge cu maxim 3 linii mai sus (de aceea eu am afișat mereu baza în care am calculat), vom verifica dacă numărul este în aceeași bază cu cea din calculator, dacă nu, vom face modificările necesare, apoi vom trimite numărul către activitatea principală, aşa cum se poate vedea în Fig. 2.5.

```

//functie pentru a extrage numarul de pe linie(primul numar) si a putea sa il trimitem catre
private fun extractFirstNumber(line: String): String {
    //verificam daca linia incepe cu un numar sau un caracter valid (A-F), altfel am apasat pe linia cu "selec baza x"
    if(line.isNotEmpty() && (line.first().isDigit() || line.first().toUppercase() in 'A' .. 'F')) {
        //luam doar primul numar de pe linie (in caz ca cineva vrea sa ia si numarul care e rezultat
        var number = ""
        for(char in line) {
            if(char.isDigit() || char.toUppercase() in 'A' .. 'F')
                number += char;
            else
                break
        }
        return number
    }
    return "" //returneaza un sir gol daca nu este un numar valid (adica daca apasam pe o linie de selectare a bazei)
}

```

Fig. 2.4 Funcție pentru extragerea numărului de pe o linie din istoric

```

//adaugam un onClickListener pentru a detecta apasarea unei linii
History.setOnItemClickListener { _, _, position, _ ->
    //luam linia selectata din istoric
    val selectedLine = HistoryData.getAll()[position]

    //extragem primul numar din linie
    val number = extractFirstNumber(selectedLine.toString())

    //memoram baza in care se afla numarul, in cazul in care am selectat o linie pe care avem un numar
    var numberBase = 10 //initial presupunem ca baza este 10
    if(number != "") {
        for (i in 1 .. 3) {
            //daca nu ni s-au terminat linile (e aproape imposibil sa nu gasim baza inainte de a se termina linile dar ok:) )
            if(position - i >= 0) {
                val line = HistoryData.getAll()[position - i]
                if(line.contains( other: "baza 16"))
                    numberBase = 16
            }
        }
        //convertim numarul la BigInteger din baza sursa
        val bigNumber = BigInteger(number, numberBase)

        //convertim numarul
        val calculatorBase = if(MainActivity.isHexMode) 16 else 10
        val converted = if(calculatorBase == 16) bigNumber.toString(radix: 16).uppercase() else bigNumber.toString()

        //trimitem numarul convertit catre MainActivity
        val intent = Intent()
        intent.putExtra( name: "selected_number", converted)
        setResult(RESULT_OK, intent)
        finish()
    }
}

```

Fig 2.5 Selectarea, convertirea și trimiterea numărului selectat către activitatea principală

Colorarea și inserarea elementelor în istoric

Pentru a avea istoricul de sus în jos, adică operațiile vechi primele(sus), operațiile noi ultimele(jos), pur și simplu adăugam elementele la finalul ListView-ului.

```
//adaugam liniile colorate in lista
HistoryList.add(line0)
HistoryList.add(line1)
HistoryList.add(line2)
HistoryList.add(line3)
```

Pentru a colora liniile în funcție de bază am creat o listă de tipul *SpannableString* în care adăugam elemente colorându-le după caz(Fig. 2.6).

```
//selectam culoarea corecta in functie de baza in care suntem
val color = if(bs == "select baza 10") {
    context.getColor(R.color.pastel_green)
} else {
    context.getColor(R.color.pastel_pink)
}

//setam culoarea corecta
val line0 = SpannableString("$bs").apply {
setSpan(ForegroundColorSpan(color), 0, length, 0) }
val line1 = SpannableString("$nr1 $op").apply {
setSpan(ForegroundColorSpan(color), 0, length, 0) }
val line2 = SpannableString("$nr2 =").apply {
setSpan(ForegroundColorSpan(color), 0, length, 0) }
val line3 = SpannableString("$res = $nr1 $op $nr2").apply {
setSpan(ForegroundColorSpan(color), 0, length, 0) }
```

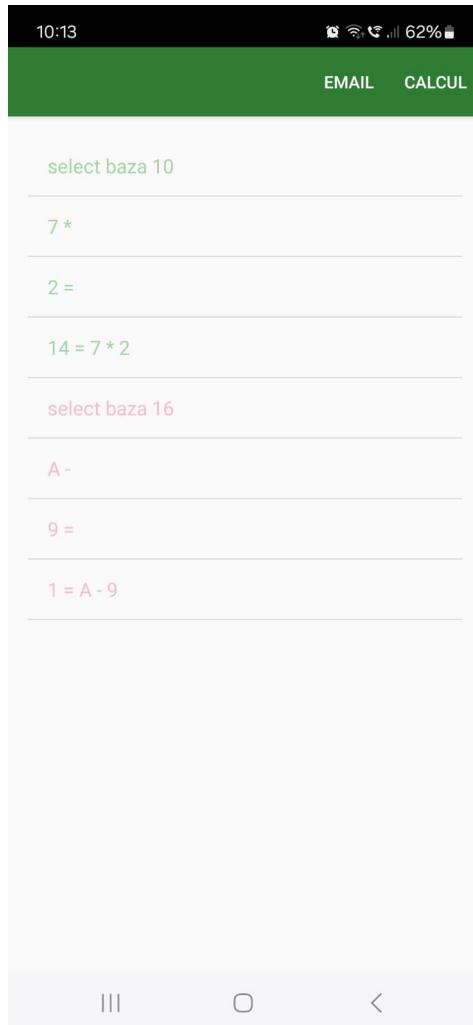


Fig. 2.6 Colorarea elementelor în istoric în funcție de bază

Bara de scroll și scroll automat la ultimele valori introduse

Bara de scroll se activează automat când textul introdus depășește mărimea ecranului, deoarece am folosit un ListView. Pentru a vedea mereu ultimele valori introduse am resetat poziția de scroll a istoricului aşa cum se poate vedea în codul de mai jos.

```
val adapter = ArrayAdapter(this,  
    android.R.layout.simple_list_item_1, HistoryData.getAll())  
History.adapter = adapter  
  
//pentru a seta bara de scroll astfel incat sa vad mereu ultimul
```

```
element  
History.setSelection(adapter.count - 1)
```

Convertirea numerelor la schimbarea bazei

Dacă la schimbarea bazei avem numere în TextA și TextC, acestea trebuie convertite. Singurul caz „mai complicat” pe care l-am întâlnit aici a fost când în TextA avem un rezultat, fapt pentru care am verificat asta și în cazul în care aveam un rezultat, după convertire adăugam și semnul egal.

```
//convertim textul din hexadecimal in decimal  
val currentText = textCurrent.text.toString()  
val anteriorText = textAnterior.text.toString()  
  
//convertim textul curent si textul anterior in noua baza  
val newTextCurrent = convertToBase(currentText, 16, 10)  
val newTextAnterior = convertToBase(anteriorText, 16, 10)  
  
//vedem daca in textAnterior avem un rezultat, adica daca avem  
semnul =, pentru ca atunci cand convertim sa il adaugam  
var e_egal = ""  
if(textAnterior.text.first() == '=')  
    e_egal = "="  
  
//Actualizam textul  
textCurrent.text = newTextCurrent  
if(e_egal == "=") {  
    textAnterior.text = "= $newTextAnterior"  
}  
else {  
    textAnterior.text = newTextAnterior  
}
```

Observații

În această secțiune aş dori să prezint câteva detalii legate de lucrurile puțin mai nesemnificative, ori despre modificările pe care eu le-am făcut aplicație pentru a îmbunătății experiența utilizatorului.

Modul Landscape

Pentru modul Landscape am creat alte layout-uri pentru ecranele 1 și 2. Am făcut câteva mici modificări în ceea ce privește mărimea textului, a butoanelor, a TextView-urilor pentru ca aplicația să arate mai bine în acest mod, fapt ce se poate vedea în Fig. 3.1.

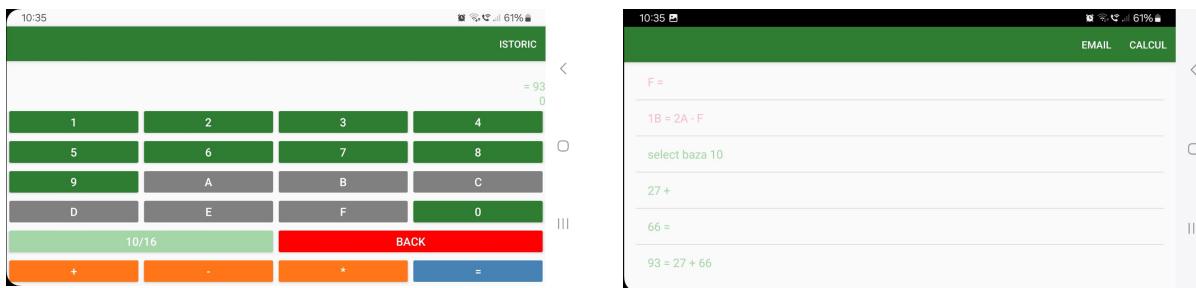


Fig. 3.1 Modul Landscape

Resetarea calculatorului când apăs pe 0-9, A-F; limita numerelor

Eu am decis că dacă în TextA am un rezultat și apăs pe una din tastele de la 0 la 9 sau de la A la F, textul din TextA să fie resetat cu 0, iar în TextC să apară tasta selectată pentru a permite utilizatorului să efectueze o nouă operație fără a fi nevoie să repornească aplicația. În cazul în care dorește să continue calculul folosindu-se de rezultatul obținut anterior, pur și simplu apasă mai întâi pe una din tastele cu operatori.

O altă modificare pe care am adus-o aplicației este limita de caractere pe care le putem introduce pentru un număr(15 – baza 10, 12 – baza 16), fapt ce îmi asigură că numerele vor fi scrise doar pe o linie și că nu voi avea erori prea mari la calcularea numerelor, pentru că nu îi permit utilizatorului să introducă numere mari.

Modificare operatorilor

Nu am înțeles foarte bine cerința ce vizează modificare operatorilor, aşa că eu am permis schimbarea lor și când avem număr în ambele câmpuri de text. Mi s-a părut și mai interesantă această abordare, permitându-i utilizatorului să schimbe operatorul oricând dorește atâtă timp cât există un rezultat sau un număr cu un operator în TextA.