UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA CALCULATOARE

PROIECT

BENCHMARK PENTRU SMARTPHONE

Disciplina : Structura Sistemelor de Calcul

Student : Pojar Andrei-Gabriel

An : III

Grupa : 30233

Profesor coordonator : Mădălin Neagu

CUPRINS

1. Introducere
    - Context
    - Motivație
    - Obiective
    - Limbajul în care e implementată aplicația
2. Studiu bibliografic
3. Analiză
4. Design
5. Implementare
6. Testare
7. Concluzii

1. Introducere

Benchmarking-ul este o metodă utilizată pentru evaluarea caracteristicilor de performanță software sau hardware ale calculatoarelor. În acest context, termenul "benchmark" are o dublă semnificație. Acesta se referă la executarea unui set de programe pentru a evalua performanța hardware-ului sau a software-ului. De asemenea, se referă și la aplicația utilizată în scopul evaluării comparative. Calitățile unui benchmark de calitate sunt: informația să fie relevantă, rezultatele măsurării să nu fie influențate de vânzător/utilizator, rezultatele să fie portabile și repetabile.

• Context

În era tehnologiei mobile, dispozitivele inteligente, în special telefoanele mobile, au devenit componente esențiale ale vieții noastre de zi cu zi. Aceste dispozitive au evoluat semnificativ în ultimii ani, de la simplele telefoane la adevarate telefoane inteligente. Cu toate acestea, o problemă comună cu care se confruntă utilizatorii este evaluarea performanței acestor dispozitive.

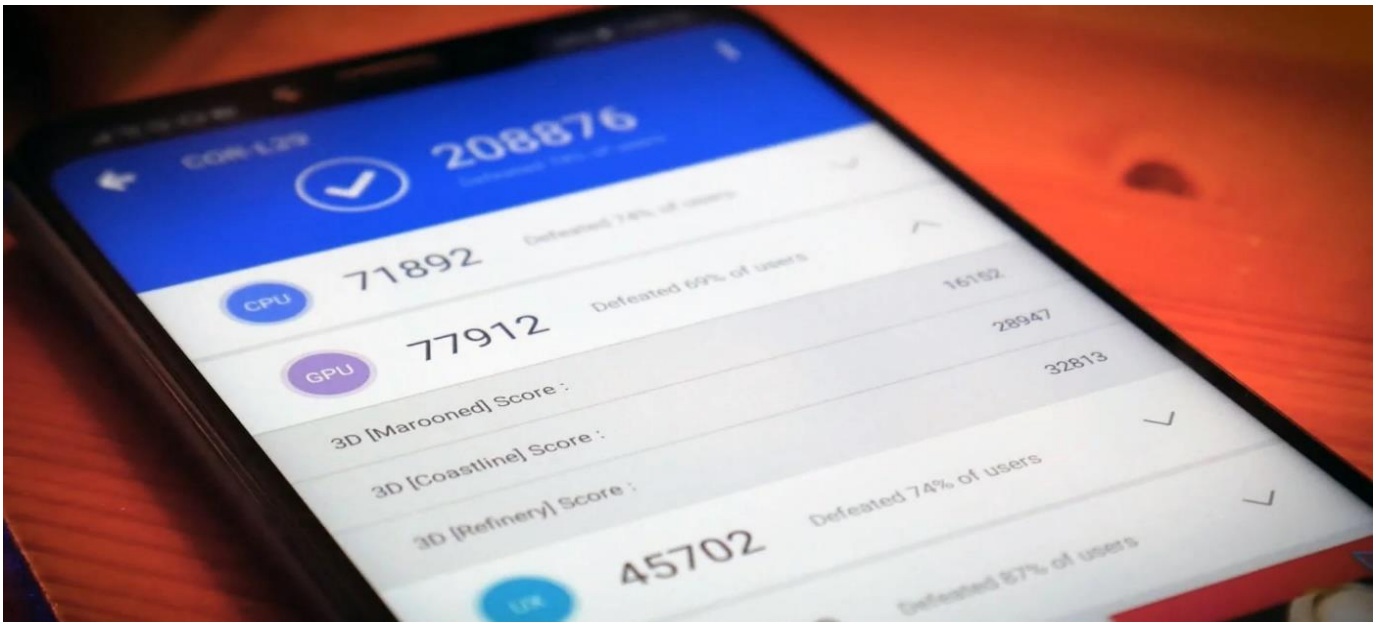
Proiectul meu propune dezvoltarea unui benchmark pentru telefoane mobile, care să permită utilizatorilor să evalueze performanța dispozitivelor lor în mod obiectiv. Acest benchmark va fi conceput pentru a testa trei aspecte cruciale ale performanței unui telefon mobil: puterea de procesare, memoria și performanța grafică (GPU). Prin intermediul acestui benchmark, utilizatorii vor putea compara performanța telefoanelor lor cu cea a altor modele disponibile pe piață.

• Motivație

Motivația acestui mic proiect de benchmark pentru telefoane mobile se bazează pe necesitatea furnizării unui instrument accesibil și ușor de utilizat pentru evaluarea performanței dispozitivelor mobile. Aceasta are la bază următoarele considerente:

- utilitate generala: Un benchmark simplu pentru telefoane mobile este util atât pentru utilizatori, cât și pentru dezvoltatori. Utilizatorii pot evalua rapid și obiectiv performanța telefoanelor lor, în timp ce dezvoltatorii pot folosi aceste informații pentru a optimiza aplicațiile lor pentru o varietate de dispozitive;
- simplificarea procesului de evaluare: Alegerea unui dispozitiv potrivit poate fi copleșitoare, având în vedere numărul mare de modele de telefoane de pe piață. Acest benchmark va simplifica procesul de evaluare, permițând utilizatorilor să compare rapid performanța dispozitivelor și să ia decizii mai înțelepte de cumpărare care să îi mulțumească pe termen lung;

- evaluarea performanței pe termen scurt: Benchmark-ul va oferi o imagine imediată a performanței dispozitivului, fără a implica evaluări complexe sau lungi sesiuni de testare, iar acest lucru va economisi timp și efort utilizatorilor;
- evaluarea evoluției dispozitivelor: Acest benchmark va permite utilizatorilor să urmărească îmbunătățirile sau declinurile performanței dispozitivului lor pe măsură ce aceste e folosit de către utilizator și să vadă dacă pe viitor mai merită să investească în acea marcă de telefon.



Prin dezvoltarea acestui benchmark simplu, îmi propun să vin în întâmpinarea nevoilor utilizatorilor de a evalua performanța dispozitivelor mobile pe care le dețin într-un mod eficient, benefic, rapid și ușor de înțeles, contribuind astfel la îmbunătățirea experienței utilizatorilor de telefoane mobile.

• Obiective

Am stabilit următoarele obiective pentru proiectul meu:

- obiectivul principal al proiectului este să dezvolt un benchmark care să permită utilizatorilor să evalueze performanța telefoanelor mobile într-un mod obiectiv și ușor de înțeles;
- măsurarea puterii de procesare
- evaluarea memoriei
- măsurarea performanței grafice (GPU)

- **Limbajul în care e implementată aplicația**

  Codul pentru aplicația Benchmark este implementat în limbajul de programare Kotlin, care este un limbaj foarte asemănător cu Java. Limbajul Kotlin diferă foarte puțin de Java, având câteva aspecte specific cum ar fi: type inference(Kotlin permite inferența automată a tipului variabilelor, expresii String interpolate (Kotlin oferă un mod concis de a formata șiruri de caractere folosind expresii interpolate), for loop range (diferă puțin cum se scrie bucla for), utilizarea lateinit (Kotlin permite amânarea inițializării variabilelor, specificarea tipului returnat de o funcție(Kotlin permite specificarea tipului de returnare pentru o funcție explicit după semnătura funcției), etc.

2. Studiu bibliografic

Bibliografie:

- https://www.cpuid.com/softwares/cpu-z.html
- https://www.youtube.com/watch?v=Wp6KbJcnxGU
- https://www.androidauthority.com/what-is-benchmarking-3299981/
- https://kotlinlang.org/docs/basic-syntax.html
- https://developer.android.com/reference/android/animation/ValueAnimator
- https://stackoverflow.com/questions/33870408/android-how-to-use-valueanimator

În ultimii ani, telefoanele mobile au evoluat semnificativ devenind dispozitive complexe cu o gamă largă de funcționalități. Această evoluție a dus la necesitatea evaluării și comparării performanței acestora. Benchmarking-ul pentru telefoane mobile a devenit o practică obișnuită pentru a oferi utilizatorilor și dezvoltatorilor o modalitate obiectivă de a evalua performanța hardware și software. Acest domeniu a început cu benchmark-uri simple pentru viteza procesorului și performanța grafică, precum Quadrant sau Linpack și s-au dezvoltat ulterior pentru a acoperi o gamă mai largă de caracteristici cum ar fi: durata bateriei, performanța camerei, gestionarea memoriei, etc.

În prezent, există o varietate de benchmark-uri utilizate pentru a evalua perfomanța telefoanelor mobile: Antutu, Geekbench, 3Dmark și GFXBench și altele.

Rezultatele benchmark-urilor au devenit un instrument important pentru a evalua performanța dispozitivelor și pentru a ajuta la luarea deciziilor de achiziție. Acest lucru a stimulat competiția în industrie și a determinat producătorii să se concentreze pe îmbunătățirea hardware-ului și software-ului.

Pe măsură ce telefoanele mobile continuă să se dezvolte se anticipează că benchmarking-ul va rămâne un aspect esențial pentru evaluarea performanței. Tendințele viitoare includ extinderea evaluării la noile aspecte ale experienței mobile cum ar fi inteligență artificială, securitatea, etc.

3. Analiză

- ## Context și relevanță

Proiectul de benchmark pentru smartphone abordează nevoia utilizatorilor și dezvoltatorilor de a evalua eficient și obiectiv performanța dispozitivelor mobile. Concentrându-se pe CPU,GPU și memorie, acesta simplifica procesul de evaluare și comparare a performanțelor.

- ## Analiza procesorului(CPU)

Procesul de analiză a performanțelor procesorului implică evaluarea timpului necesar pentru a efectua sarcini de procesare intensivă. Clasa CpuActivity include o simulare a sarcinii de benchmark, măsurând timpul total de execuție. Această analiză oferă o perspectivă asupra capacităților de calcul ale procesorului și permite identificarea potențialelor zone de îmbunătățire.

- ## Analiza procesorului grafic(GPU)

Analiza procesorului grafic este realizată prin intermediul clasei GpuActivity care utilizează o animație continuă pentru a evalua ratele de cadre pe secunda(FPS). Prin măsurarea FPS-ului se obține o imagine asupra performanțelor GPU-ului în tratarea graficelor și animațiilor.

- **Analiza memoriei**

Analiza performanțelor memoriei este realizată în clasa MemoryActivity. Se colectează informații despre utilizarea memoriei în timpul unei sarcini intensive, iar rezultatele includ memoria inițială, memoria disponibilă și detalii despre sistemul de operare. Această analiză furnizează informații esențiale despre modul în care dispozitivul gestionează memoria.

- **Metrici de performanță**

Pentru a evalua performanța dispozitivului mobil în contextul benchmarking-ului este esențial să definim și să utilizăm metrici de performanță relevante pentru fiecare componentă.

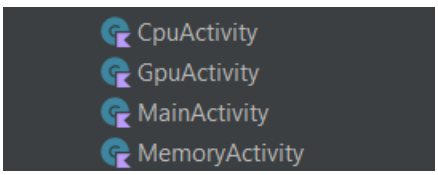
Pentru procesor timpul de execuție reprezintă durata totală necesară pentru a efectua sarcina de procesare. Cu cât timpul este mai mic, cu atât performanța procesorului este mai bună.

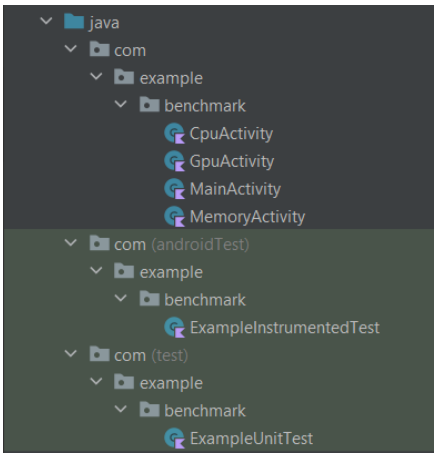
Pentru memorie timpul de execuție pentru sarcina intensivă de memorare indică durata totală pentru a efectua sarcina intensivă de memorie. Mai trebuie să avem in vedere la memorie și cantitatea de memorie ocupată inițial de aplicație, cantitatea de memorie utilizată în timpul sarcinii și capacitatea totală, respectiv disponibilitatea memoriei pe dispozitiv.

Pentru procesorul grafic ratele de cadre pe secundă furnizează o măsură a performanțelor grafice ale procesorului grafic. Un număr mai mare de cadre pe secundă indică o mai bună gestionare a sarcinilor grafice.

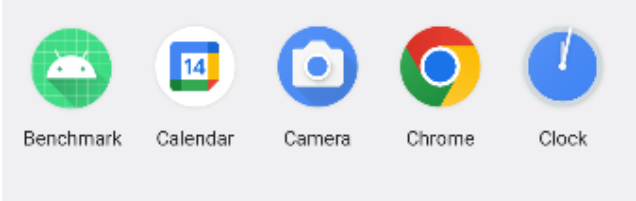
4. Design

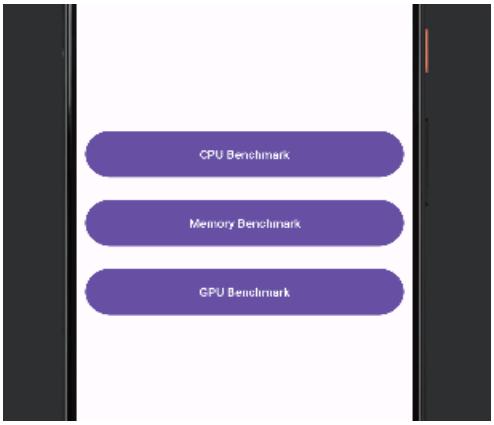
Designul proiectului este cuprinzător și intuitiv și acoperă evaluarea procesorului, a memoriei și a unității grafice, asigurând o analiză completă și obiectivă. Proiectul este structurat în mai multe module, fiecare gestionând evaluarea performanței pentru un aspect specific. Fiecare evaluare (CPU, GPU, memorie) este implementată într-o activitate dedicată (CpuActivity, GpuActivity, MemoryActivity). Avem și module pentru teste(test unitar și test de instrumentare).





Interfața principală are trei butoane fiecare pentru evaluarea performanței pentru un aspect. Pentru a ajunge în această fază va trebui să rulezi aplicația să te duci în meniul telefonului și să dai click pe aplicația Android numită Benchmark. După ce s-a dat click pe aplicație se va deschide această fereastră în care ai trei butoane fiecare cu funcționalitatea lui.





La click pe butonul de evaluarea a procesorului se va deschide o noua fereastră unde vom putea vizualiza CPU Benchmark Time, la click pe butonul de evaluarea a memoriei vom putea vizualiza ceva mai mult, adică: Memory Benchmark Time, Initial Memory Usage, Memory Usage durin task, Total Ram, Available Ram, Android Version, Result. La click pe butonul de evaluare a unității grafice se va afișa FPS-ul(numărul de cadre pe secundă).

De asemenea, avem și fișierele .xml care descriu aspectul vizual al interfeței utilizatorului pentru activitățile în cadrul aplicației Android. Fișierul activity_cpu.xml definește structura vizuală a activității CPU. În cadrul acestei activități principalul element vizual este un TextView cu ID-ul tvResult care va afișa timpul de execuție al benchmark-ului. Cu ajutorul lui ConstraintLayout se organizează și se controlează poziționarea elementelor în interfața utilizatorului. Fișierul activity_gpu.xml este exact ca activity_cpu.xml doar că apare în plus și o animație. Cu ajutorul android:src specificăm sursa imaginii, în cazul nostru o săgeata și o setăm să fie deasupra rezultatului. Fișierul activity_memory.xml este la fel ca activity_cpu.xml. În fișierul activity_main.xml reprezentăm ecranul principal al aplicației. Tot cu ajutorul unui ConstraintLayout imbricat în interiorul layout-ului rădăcină organizăm cele trei butoane aflate pe pagina principală a aplicației. Organizăm aceste trei butoane astfel încât să se situeze unul sub altul să ofere o pagină principală cât mai simplă și ușor de utilizat.

Proiectul utilizează eficient resursele dispozitivului, evitând supraîncărcarea inutilă a memoriei. Implementarea evaluărilor este optimizată pentru a furniza rezultate rapide, minimizând timpul de așteptare al utilizatorului. Designul proiectului de benchmark pentru smartphone se axează pe claritate, eficiență și ușurință de utilizare.

5. Implementare

Implementarea proiectului de benchmark pentru telefon mobil este realizată în limbajul de programare Kotlin în mediul de dezvoltare Android Studio. Codul este structurat în mod clar și eficient, respectând cele mai bune practici de programare Android. Proiectul este organizat în module distincte pentru evaluarea CPU, GPU și memorie, asigurând un cod ușor de gestionat. Clasele Kotlin pentru calcularea benchmark-ului pentru fiecare dintre procesor,unitate grafică și memorie:

- CpuActivity – Metoda simulateCPUBenchmarkTask(): Această metodă simulează o sarcină intensivă pentru CPU prin intermediul unui ciclu de calcul matematic (for (i in 0 until 1000000000)). Această metodă simulează o sarcină de benchmark CPU prin efectuarea unei bucle for care efectuează o serie de operații simple de calcul. Se măsoară timpul de început și sfârșit al sarcinii de benchmark utilizând System.currentTimeMillis() și returnează durata totală a sarcinii. Metoda setData(): În această metodă, rezultatul obținut din simularea benchmark-ului CPU este afișat într-un element TextView (tvResult) în formatul "CPU Benchmark Time: $elapsedTime ms".

```
    private fun setData()
    {
        val elapsedTime=simulateCPUBenchmarkTask()
        val resultCPUBenchmark="CPU Benchmark Time: $elapsedTime ms"
        result.text=resultCPUBenchmark
    }
    private fun simulateCPUBenchmarkTask():Long
    {
        val startTime=System.currentTimeMillis()
        //se efectueaza o bucla mare pt o operatie de multiplicare
        for(i in 0 ≤ until < 1000000000)
        {
            val result=i*2
        }
        val endTime=System.currentTimeMillis()
        return endTime-startTime
    }
}
```

- GpuActivity - Metoda setupAnimation(): Această metodă efectuează o animație pe GPU și calculează numărul de cadre pe secundă(FPS) pentru a evalua performanța animației. Cu ajutorul metodei setupAnimation se creează și configurează un obiect ValueAnimator care realizează animația pe axa X pentru imageView. Se stabilește durata, modul de repetare și numărul de repetări ale animației. Cu animator.addUpdateListener adaug un ascultător la obiectul ValueAnimator. Un ValueAnimator este utilizat pentru a gestiona animația și are capacitatea de a notifica utilizatorii atunci când se actualizează valorile sale. La fiecare cadru al animației se incrementează o variabilă care urmărește să calculeze numărul total de cadre desenate și va fi utilizat ulterior pentru calculul FPS-ului. Apoi cu imageView.translationX=it.animatedValue as Float controlez translația pe axa X a imaginii. Prin actualizarea ei, imaginea se mișcă pe ecran în funcție de valoarea animată. După aceea, cu ajutorul metodei invalidate() forțez redesenarea obiectului. Aceasta se folosește pentru a declanșa desenarea imaginii la fiecare cadru al animației, asigurându-se că schimbările sunt vizibile pe ecran. Pe scurt această parte se ocupă de actualizarea contorului de cadre, de modificarea poziției imageView pe axa X în funcție de valoarea animată și de invalidarea imageView pentru a declanșa redesenarea acesteia la fiecare cadru al animației. Metoda calculateFps(): Această metodă calculează cadrele pe secundă pentru a evalua performanța animației. Aici se resetează valorile pentru a începe un nou ciclu de calculare FPS. Programarea metodei calculateFps() pentru a fi apelată din nou după o secundă folosind imageView.postDelayed(). Pe scurt, această metodă realizează o animație pe GPU și calculează FPS-ul pentru a evalua cât de fluentă este

animația. Valorile FPS-ului sunt actualizate într-un TextView la fiecare secundă, iar animația se repetă în mod continuu. Acest tip de măsurare a FPS-ului este util în dezvoltarea de aplicații mobile pentru a asigura o experiență utilizator fluidă.

```
private fun setupAnimation() {
    // configurare animatie-set up
    val animator = ValueAnimator.ofFloat( ...values: 0f, 500f)
    animator.duration = 1000 // Durata animatiei in milisecunde
    animator.repeatMode = ValueAnimator.REVERSE
    animator.repeatCount = ValueAnimator.INFINITE

    animator.addUpdateListener { it: ValueAnimator
        // Actualizarea numarului de cadre pentru fiecare cadru de animatie
        frameCount++

        // Invalidarea imageView la fiecare cadru pentru a declansa o redesenare
        imageView.translationX = it.animatedValue as Float
        imageView.invalidate()
    }
    animator.start()
    // Start time pentru calculul FPS
    startTime = System.nanoTime()
    // Programarea(schedule) calculului FPS la fiecare secunda
    calculateFps()
}
```

```
private fun calculateFps() {
    // Calculare FPS in fiecare secunda
    val currentTime = System.nanoTime()
    val elapsedTime = currentTime - startTime
    val fps = (frameCount * 1e9f)/ elapsedTime

    result.text = "FPS: $fps"
    // Resetarea valorilor pentru urmatoarea secunda
    frameCount = 0
    startTime = currentTime

    // Programarea(schedule) a urmatorului calcul FPS dupa o secunda
    imageView.postDelayed({ calculateFps() }, delayMillis: 1000)
}
```

- MemoryActivity – Metoda handleMemory(): Această metodă colectează informații legate de utilizarea memoriei, precum utilizarea inițială a memoriei, memoria totală și disponibilă, iar apoi simulează o sarcină intensivă de memorie prin intermediul metodei simulateMemoryBenchmarkTask() și înregistrează timpul de început și sfârșit al acestei sarcini. Metoda simulateMemoryBenchmarkTask(): Simulează o sarcină intensivă de memorie prin crearea și manipularea unui array mare. În acest caz se aloca un array de Int-uri și se efectuează operații simple de scriere pentru a simula utilizarea memoriei.

Se înregistrează timpul de început și sfârșit al acestei sarcini și se returnează durata totală a benchmark-ului. Informațiile obținute sunt apoi afișate într-un element TextView (tvResult) sub forma unui raport detaliat care include timpul de execuție, utilizarea inițială a memoriei, memoria disponibilă, versiunea Android și nivelul API.

```
private fun handleMemory()
{
    val startTime=System.currentTimeMillis()
    // Informatii legate de memorie
    val initialMemoryUsage = getMemoryUsage()
    // in MB
    val totalMemory = Runtime.getRuntime().maxMemory() / (1024 * 1024)
    val availableMemory = Runtime.getRuntime().freeMemory() / (1024 * 1024)

    // Simulam o sarcina cu utilizare intensiva a memoriei
    val resultMemoryBenchmark = simulateMemoryBenchmarkTask()

    val endTime = System.currentTimeMillis()
    val elapsedTime = endTime - startTime

    // Android version + API level
    val androidVersion = Build.VERSION.RELEASE
    val apiLevel = Build.VERSION.SDK_INT

    val resultData = "Memory Benchmark Time: $elapsedTime ms\n" +
            "Initial Memory Usage: $initialMemoryUsage MB\n" +
            "Memory Usage During Task: ${initialMemoryUsage - availableMemory} MB\n" +
            "Total RAM: $totalMemory MB\n" +
            "Available RAM: $availableMemory MB\n" +
            "Android Version: $androidVersion (API $apiLevel)\n" +
            "Result: $resultMemoryBenchmark"
    result.text = resultData
}
```

```
// Simulam o sarcina cu utilizare intensiva a memoriei
private fun simulateMemoryBenchmarkTask(): Long {
    val startTime = System.currentTimeMillis()
    val array = IntArray( size: 10000000)
    for (i in 0 ≤ until < 10000000) {
        // Simularea operatiilor de memorie
        array[i] = i
    }
    val endTime = System.currentTimeMillis()
    return endTime - startTime
}
```

Meniul principal(MainActivity):

- MainActivity – reprezintă activitatea principală a aplicației mele și servește ca punct de intrare pentru utilizatori. Scopul său principal este de a oferi o interfață utilizator simplă pentru a accesa și efectua benchmark-uri pe diferite aspecte ale

dispozitivului. În clasa MainActivity, am creat un meniu principal care conține trei butoane pentru a accesa fiecare activitate de benchmarking. Fiecare buton este asociat cu activitatea corespunzătoare cu ajutorul metodei handleClick() care asociază fiecărui buton din meniul principal (cpuBenchmarkButton, memoryBenchmarkButton, gpuBenchmarkButton) acțiunea de a deschide activitatea corespunzătoare când butonul este apăsat. Putem observa că este utilizată o expresie specifică limbajului Kotlin: CpuActivity::class.java care obține o referință la obiectul Class asociat clasei CpuActivity. Această sintaxă face parte din API-ul Kotlin și este folosită pentru a accesa metadatele asociate unei clase în timpul execuției programului.

```
class MainActivity : AppCompatActivity() {
    private lateinit var cpuBenchmarkButton: Button
    private lateinit var memoryBenchmarkButton: Button
    private lateinit var gpuBenchmarkButton: Button
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        initData()
    }
    private fun initData()
    {
        cpuBenchmarkButton=findViewById(R.id.cpuBenchmarkButton)
        memoryBenchmarkButton=findViewById(R.id.memoryBenchmarkButton)
        gpuBenchmarkButton=findViewById(R.id.gpuBenchmarkButton)
        handleClick()
    }
    private fun handleClick()
    {
        cpuBenchmarkButton.setOnClickListener{ it: View!
            startActivity(Intent( packageContext: this,CpuActivity::class.java))
        }
        memoryBenchmarkButton.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, MemoryActivity::class.java))
        }
        gpuBenchmarkButton.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, GpuActivity::class.java))
        }
    }
}
```

Teste unitare și de instrumentare:

- ExampleUnitTest - Acesta este un exemplu de test unitar care verifică dacă adunarea a doi întregi este corectă. Puteți extinde aceste teste pentru a verifica și alte funcționalități ale aplicației.
- ExampleInstrumentedTest - Acesta este un exemplu de test de instrumentare care verifică contextul aplicației. Acest test utilizează framework-ul de testare Android pentru a obține contextul și a se asigura că pachetul corespunde celui așteptat.

6. Testare ->Validare

Testarea și validarea reprezintă procese esențiale pentru asigurarea calității și corectitudinii unei aplicații.

- ExampleUnitTest

Această clasă conține teste unitare simple pentru a verifica funcționalități de bază. Validare: asigură că operațiile de bază sunt implementate corect în aplicație.

```
package com.example.benchmark

import ...

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
//testare unitara
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals( expected: 4,  actual: 2 + 2)
    }
}
```

- ExampleInstrumentedTest

Acest test utilizează framework-ul de testare Android pentru a valida contextul aplicației. Validare: verifică dacă contextul aplicației corespunde așteptărilor, asigurându-se că aplicația este configurată corect în cadrul mediului Android.

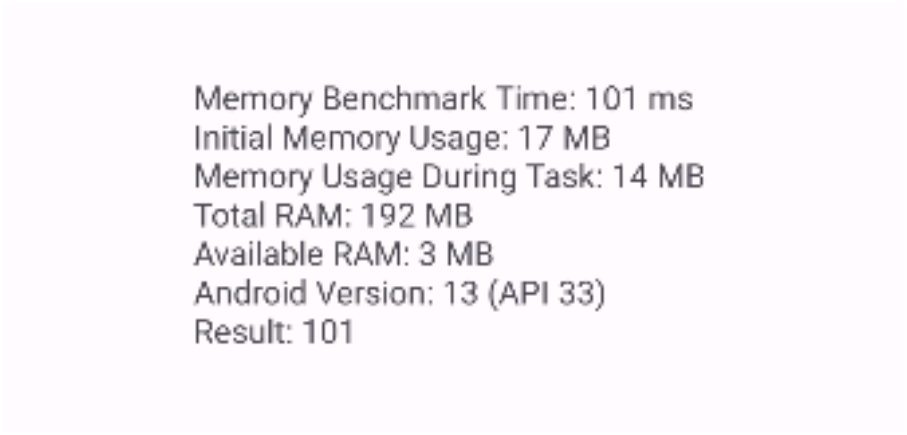
```
package com.example.benchmark

import ...

/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
//test de instrumentare
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals( expected: "com.example.benchmark", appContext.packageName)
    }
}
```

- Testare pentru componentele specifice
    - Testare pentru procesor

    - Testare pentru memorie



    - Testare pentru procesor grafic

Pentru procesor îmi afișează în ms timpul de execuție care reprezintă durata totală necesară pentru a efectua sarcina de procesare. Pentru memorie se afișează timpul de execuție pentru sarcina de memorare în ms, cantitatea de memorie ocupată inițial de aplicație și câtă memorie este folosită în timpul sarcinii, capacitatea totală a memoriei și câtă memorie RAM mai este disponibilă după ce s-a realizat sarcina, respectiv versiunea de Android și timpul de execuție al simulării sarcinii de memorie. Pentru procesorul grafic se afișează numărul de cadre pe secundă a animației care simulează pe ecran.

7. Concluzii

Proiectul meu de benchmarking pentru smartphone furnizează o evaluare utilă a performanțelor componentelor cheie ale dispozitivului: procesor, memorie și unitate grafică. În urma implementării și testării s-au obținut rezultate semnificative care furnizează informații esențiale despre performanța componentelor. Pentru procesor timpul de execuție reprezintă durata totală necesară pentru a efectua sarcina de procesare. Pentru memorie timpul de execuție pentru sarcina intensivă de memorare indică durata totală pentru a efectua sarcina intensivă de memorie, iar pentru procesorul grafic ratele de cadre pe secundă furnizează o măsură a performanțelor grafice ale procesorului grafic.

Potențialii dezvoltatori pot utiliza concluziile acestui proiect de benchmark pentru optimizări ulterioare și îmbunătățirea experienței utilizatorului în aplicații viitoare. O posibilă optimizare ar fi integrarea de teste suplimentare pentru a acoperi o gamă mai largă de scenarii de utilizare. O altă posibilă optimizare și dezvoltare ar fi monitorizarea continuă a performanțelor pe măsură ce apar noi actualizări de hardware și software.

Proiectul poate fi extins pentru a acoperi și evalua performanțele pe diferite platforme sau dispozitive, cum ar fi tablete, smartwatch-uri sau alte tipuri de smartphone-uri mai performante. O altă direcție de extindere ar fi adaptarea proiectului pentru a funcționa pe dispozitive cu sisteme de operare diferite, precum iOS, oferind astfel dezvoltatorilor și utilizatorilor oportunitatea de a compara performanțele între diferite platforme.