Energetska učinkovitost Android aplikacija

Radojčić, Hana

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

ENERGETSKA UČINKOVITOST ANDROID APLIKACIJA

Rijeka, srpanj 2023.

Hana Radojčić
0069078178

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

ENERGETSKA UČINKOVITOST ANDROID APLIKACIJA

Mentor: izv. prof. dr. sc. Sandi Ljubić

Rijeka, srpanj 2023.

Hana Radojčić
0069078178

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 3. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj mobilnih aplikacija**
Grana: **2.09.01 arhitektura računalnih sustava**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Hana Radojčić (0069078178)**
Studij: Diplomski sveučilišni studij računarstva
Modul: Računalni sustavi

Zadatak: **Energetska učinkovitost Android aplikacija / Energy Efficiency of Android Applications**

Opis zadatka:

U sklopu diplomskog rada potrebno je analizirati utjecaj pristupa razvoju mobilnih aplikacija za Android OS na potrošnju resursa, od čega u primarni fokus treba staviti energetsku učinkovitost. Upoznati se sa postojećim smjernicama za izgradnju energetski učinkovitih Android aplikacija. Detektirati i analizirati uobičajene anti-oblikovne-obrasce (engl. anti-patterns) koji negativno utječu na potrošnju baterije mobilnog uređaja. Demonstrirati kako se energetska učinkovitost Android aplikacija može poboljšati postupcima refaktoriranja te argumentirati odgovarajuće intervencije u programskom kodu.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Sandi Ljubić

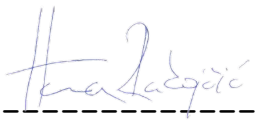
Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad uz upotrebu navedene literature i dokumentacije korištenih tehnologija.

Rijeka, srpanj 2023.



Hana Radojčić

Zahvala

Zahvaljujem svom mentoru prof. dr. sc. Sandiju Ljubiću na razumijevanju, savjetima i pruženoj pomoći prilikom izrade ovog rada.

Hvala obitelji, prijateljima i kolegama na podršci tijekom mog akademskog putovanja.

Sadržaj

Popis slika

Popis tablica

Popis isječaka programskog koda

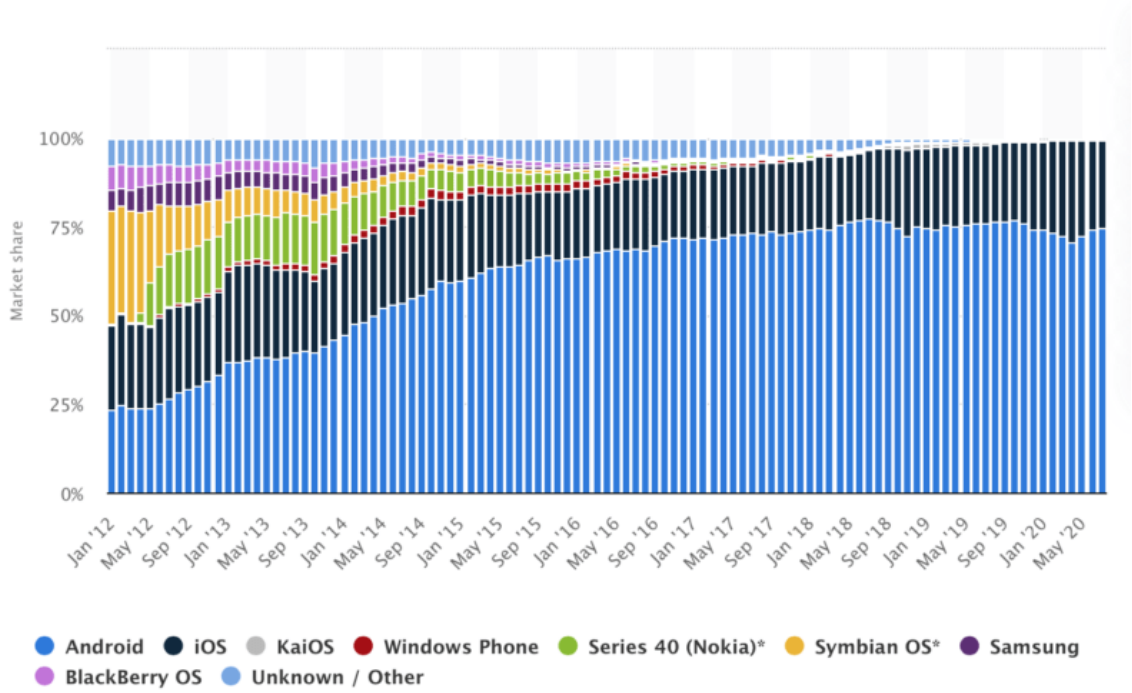
Poglavlje 1

Uvod

Svakim danom, mobilna tehnologija sve više napreduje te se mobilni i nosivi uređaji sve više infiltriraju u naš svakodnevni život. Sve većom uporabom takvih uređaja, dolazi do potražnje za što boljim izvršavanjem zadataka. To se može očitovati u bržem izvođenju procesa na uređaju uz minimalno zauzeće resursa te minimalnu potrošenu energiju. Budući da se već godinama svijetom širi ideja o ekološki osviještenom životu, shodno tome, mobilni uređaji također trebaju biti energetski učinkoviti. Iako sama izvedba mobilnog uređaja ovisi o proizvođaču, razvijene i instalirane aplikacije od trećih strana uvelike utječu na potrošnju energije uređaja. Nepotrebna potrošnja baterije, osim ekološkog utjecaja, može istovremeno utjecati i na korisničko iskustvo, stoga je važno da tvrtke informatičkih tehnologija isporuče učinkovite mobilne aplikacije.

Današnje tržište mobilnih operacijskih sustava se većinom sastoji od iOS i Android operacijskih sustava. Na slici 1.1. se može vidjeti tržišni udio mobilnih operacijskih sustava u posljednjih deset godina. Iz toga se može zaključiti da trenutačno Android mobilni uređaji prevladavaju tržištem s 71.47% udjela u tržištu, dok iOS ima 27.88% udjela, a ostatak od 0.65% udjela se odnosi na ostale operacijske sustave [1].

Budući da se uporaba Android operacijskog sustava proširuje i na ostale pametne uređaje poput televizora, satova i automobila, potrebno je znati kako napraviti energetski učinkovitu Android aplikaciju. U ovome radu će se prvenstveno razmatrati



Slika 1.1 Tržišni udio mobilnih operacijskih sustava u svijetu od siječnja 2012. do kolovoza 2022 [1]

energetska učinkovitost Android aplikacije na mobilnom uređaju, direktno mjereći parametre baterije tijekom izvođenja testiranja unutar izrađene aplikacije. Postoje određene smjernice za optimizaciju izvedbe koju je Google objavio u svojim dokumentima za razvoj Android aplikacije te su istraživači već dotakli ovu temu u pojedinim dijelovima. Ovim radom će se ispitati njihovi zaključci te ih ujedno proširiti vlastitim zaključcima. Kao rezultat se očekuje skup smjernica za optimizaciju izvedbe Android aplikacija.

Poglavlje 2

Opis tehnologija

Princip testiranja energetske učinkovitosti je implementiran Android aplikacijom izrađenom u Android Studiju s programskim jezikom Kotlin. Ona se u principu sastoji od dva dijela. Sadrži promjenjivi dio koda koji će se mijenjati ovisno o potrebi testiranja te dio koda koji pokreće testiranje i direktno analizira svojstva baterije za vrijeme testiranja pomoću razreda *BatteryManager*. Testiranja se izvode na fizičkom uređaju, preciznije mobilnom uređaju marke Huawei.

2.1 Android Studio

Android Studio je službeno integrirano razvojno okruženje za razvoj Android aplikacija razvijen od strane Google-a. Omogućava razne značajke pomoću kojih pruža optimiziran razvoj kvalitetnih aplikacija za svaki Android uređaj. Najavljen je 2013. godine, a prva stabilna verzija je objavljena 2014. godine. Kotlin je preferirani programski jezik za razvoj aplikacija, ali Android studio podržava i druge programske jezike poput Java, C++ i JavaScript. U ovom slučaju, za razvoj Android aplikacije je korištena verzija Android Studio Dolphin koja je objavljena u rujnu 2022. godine [2] [3].

2.2 Kotlin

Kotlin je moderni više platformski programski jezik kojeg Google preporučuje za razvoj aplikacija namijenjene Android operacijskom sustavu. Kompatibilan je s Javom, a u usporedbi s njom, Kotlin je mnogo jednostavniji te ga koriste mnogi početnici, kao i profesionalni Android razvojni programeri. Također pomaže u povećanju produktivnosti, zadovoljstva programera i sigurnosti koda [4].

2.3 BatteryManager

BatteryManager je razred Android operacijskog sustava koja sadrži znakovne nizove i konstante koji se koriste za vrijednosti kod promjena svojstva i parametara baterije Android uređaja. Ona pruža metodu za dohvaćanje informacija o svojstvima baterije i punjenja. Stoga se s ovim razredom mogu pribaviti informacije poput trenutačnog zdravlja baterije, trenutačni postotak napunjenosti baterije kao i trenutačan napon, temperatura i izlazna struja baterije [5].

2.4 Huawei P40 Lite

Huawei P40 Lite je Android pametni telefon izdan 2020. godine. Ima 6,4-inčni LCD zaslon i radi na Android 10 operacijskom sustavu s Huaweijevim EMUI 10.0. Uređaj pokreće osmojezgreni Kirin 810 procesor i 6GB RAM-a. Ima 128 GB interne memorije. Napaja ga litij-ionska baterija od 4200 mAh i podržava brzo punjenje. Mogućnosti povezivanja na uređaju uključuju 4G LTE, Wi-Fi, Bluetooth 5.0 i NFC [6].

S obzirom na to da je uređaj star tri godine, može ga se smatrati adekvatnim za izvođenje testiranja jer je relativno novi uređaj s novijim procesorom i dobrim trajanjem baterije. Osim toga, Huawei P40 Lite je uređaj srednjeg razreda, čime se može pretpostaviti da će izvođenje Android aplikacija biti optimalno.

Kako bi se kasnije rezultati testiranja mogli lakše razumjeti, potrebno je kapacitet baterije pretvoriti iz miliamper-sati u vat-sate. Pretvorba se radi prema formuli 2.1

gdje se za električni naboj uzima 4200 mAh te prosječni napon od 3,8 V. Prema tome, dobiva se vrijednost od 15,96 Wh.

$$Energija\,(Wh) = Električni\,naboj\,(mAh) * Napon\,(V)\,/\,1000\ \ [7] \tag{2.1}$$

Poglavlje 3

Proces testiranja

Kao što je već spomenuto, testiranje potrošene energije se odvija putem izrađene aplikacije koristeći razred *BatteryManager* iz Android operacijskog sustava. U idućim potpoglavljima će se opisati rad same aplikacije te postupak provođenja testiranja.

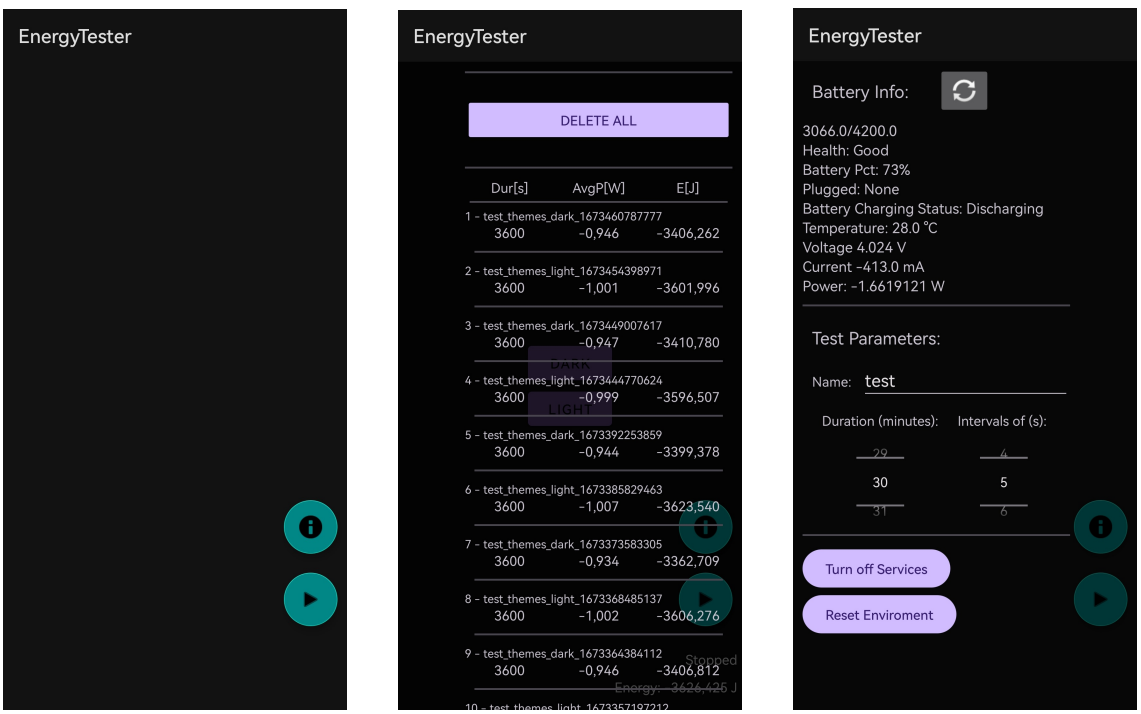
3.1 Opis aplikacije

Kako bi se što bolje i jednostavnije provelo testiranje raznih elemenata i programskih kodova Android aplikacije, razvoj aplikacije je podijeljen u dva dijela. Prvi dio se sastoji od statičkog dijela aplikacije pomoću kojeg se pokreće testiranje te uzimaju potrebni podaci za računanje potrošene energije. Drugi dio se odnosi na segment koji će se mijenjati s vremenom ovisno o testu koji se provodi. Naime, u ovom potpoglavlju će biti opisan statički dio koji je u svim provedenim testovima isti, dok će se XML datoteke i Kotlin programski kod vezani za određeni test biti opisani kod provedbe tog testa.

3.1.1 Izgled

Za početak, izgled aplikacije se može vidjeti na slici 3.1a gdje je prikazana glavna stranica na kojoj se nalaze dva gumba. Prvi gornji gumb služi za otvaranje ladica koje se mogu vidjeti slikama 3.1b i 3.1c Njima se mogu postaviti parametri testa,

vidjeti informacije vezane za bateriju te dobiti pregled svih provedenih testova. Drugi donji gumb pokreće testiranje s izabranim parametrima. Ostatak glavnog zaslona se mijenja tijekom provedbe cjelokupnog testiranja, ovisno o provedenom testu.



(a) Glavna aktivnost

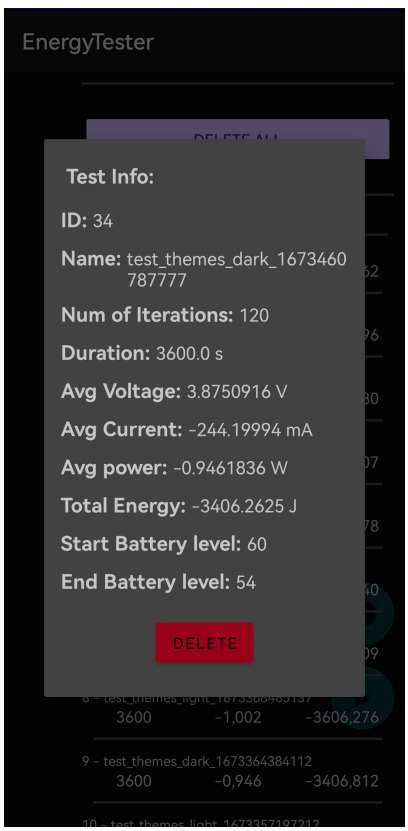
(b) Ladica za informacije o testovima

(c) Ladica za informacije o bateriji i postavljanje postavki testa

Slika 3.1 Snimke zaslona aplikacije

Jednim kratkim klikom na prvi gornji gumb, otvara se ladica s informacijama o bateriji i parametrima testa. Ovdje korisnik može vidjeti informacije poput kapaciteta baterije kao i vrijednosti struje, napona te temperature. Također, klikom na gumb pokraj naslova *Battery Info*, vrijednosti baterije će se osvježiti. Više o dohvaćanju vrijednosti baterije se nalazi u potpoglavlju 3.1.3. Ispod podataka o bateriji, nalaze se elementi kojima se dodatno mogu postaviti parametri testa poput imena i duljine trajanja. Za kraj se nalaze dva gumba kojima se mogu ugasiti servisi poput WiFi mreže, mobilne mreže i Bluetooth-a te postaviti okruženje u kojem se obavlja testiranje. Detaljnije o tim funkcionalnostima se nalazi u potpoglavlju 3.1.4

Jednim dugim klikom na prvi gornji gumb, otvara se ladica s pregledom provedenih testova. Ovdje se mogu vidjeti sve relevantne informacije spremljenih provedenih testova. Na slici 3.2 je prikazan otvoreni dialog određenog testa koji prikazuje sve spremljene informacije o testu. Također mogu se izbrisati svi testovi ili pojedini određeni test putem dijaloškog okvira.



Slika 3.2 Prikaz detalja pojedinog provedenog testa

3.1.2 Podaci

Prije opisa programskog dijela aplikacije, potrebno je navesti s kojim podacima će se upravljati. Naime, za spremanje trenutačnih vrijednosti iz baterije, točnije vrijednosti struje, napona, snage i postotka napunjenosti kao i za spremanje prosječnih vrijednosti struje, napona i snage, koriste se globalne varijable kako bi bile lako dostupne u cijelom *MainActivity* razredu i njegovim metodama. Također, ukupna

potrošena energija se zbraja u globalnoj varijabli tijekom izvođenja testa. Inicijalizaciju tih varijabli se može vidjeti na isječku 3.1. Također se mogu uočiti varijable koje se koriste za postavljanje parametara određenog testa kao što su duljina trajanja testa (*testDuration*), duljina trajanja intervala za uzimanje podataka (*testInterval*) i ime testa (*testName*). Ime testa će se postaviti programski kod izvođenja određenog testa, no, izmjena se također postiže unosom u odgovarajuće elemente sadržanim u ladici s informacijama o bateriji i parametrima testa opisanim u prethodnom potpoglavlju.

Ispis 3.1 Inicijalizacija globalnih varijabli

```
1  private var totalEnergy : Float = 0.0F
2  private var avgPowerW : Float = 0.0F
3  private var powerW : Float = 0.0F
4  private var avgCurrentMA : Float = 0.0F
5  private var currentMA : Float = 0.0F
6  private var avgVoltageV : Float = 0.0F
7  private var voltageV : Float = 0.0F
8  private var startBatLevel : Int = 0
9  private var endBatLevel : Int = 0
10
11 private var testDuration : Long = 30
12 private var testInterval : Int = 5
13 private var testName : String = "test"
14
15 private var currentTest = Test(0, testName, ArrayList(),
16                                 0, 0.0F,0.0F,0.0F,0.0F, 0.0F, 0, 0)
17 private var testList = ArrayList<Test>()
```

Uz to postoji varijabla *currentTest* razreda *Test* koja služi za spremanje izmjerenih vrijednosti. Struktura razreda *Test* se može vidjeti na isječku 3.2. Naime, kod svakog testa, podaci se uzimaju u određenom intervalu što u konačnici rezultira prosječnim vrijednostima iteracija kao završnim vrijednostima testa. Stoga, razred *Test* sadrži prosječne vrijednosti struje, napona i snage. Uz to, sadrži ime testa, sve izvedene iteracije i njihov ukupni broj, ukupnu potrošenu energiju te početnu i završnu razinu baterije. Svaki izvedeni test će se pohraniti u listu testova *testList* koji su zatim prikazani u ladici s pregledom svih provedenih testova.

Kao što je spomenuto, svaki test se sastoji od određenog broja iteracija. Stoga uz podatkovni razred *Test* s isječka 3.2, postoji podatkovni razred *Iteration* prikazan isječkom 3.3. On služi za spremanje pojedinih intervala što se može vidjeti na liniji

4 u isječku 3.2 gdje parametar *iterations* sadrži listu iteracija.

Ispis 3.2 Struktura podatkovne razreda Test

```
1
2  data class Test(var id : Long,
3                  var name : String,
4                  val iterations : ArrayList<Iteration>,
5                  var iteration_size : Int,
6                  var duration_s : Float,
7                  var avg_current_mA : Float,
8                  var avg_voltage_V : Float,
9                  var avg_power_W : Float,
10                 var energy_J : Float,
11                 var startBatLevel : Int,
12                 var endBatLevel : Int
13                 )
```

Ispis 3.3 Struktura podatkovne razreda Iteration

```
1
2  data class Iteration(val number : Int,
3                       val duration_s : Float,
4                       val current_mA : Float,
5                       val voltage_V : Float,
6                       val power_W : Float,
7                       val energy_J : Float
8                       )
```

3.1.3 Dohvaćanje vrijednosti baterije

Što se tiče programskog dijela aplikacije, prvenstveno je bilo potrebno implementirati dohvaćanje trenutačnih parametara baterije. Stoga, bilo je potrebno definirati i registrirati prijemnik emitiranja (eng. *BroadcastReceiver*), koji objavljuje vrijednosti baterije.

Kada pristignu informacije poziva se funkcija *updateBatteryData()*, gdje se uzimaju informacije poput zdravlja i kapaciteta baterije, postotka napunjenosti te vrijednosti napona, struje i temperature. Sve informacije i njihov prikaz u aplikaciji se mogu vidjeti na slici 3.1c. U isječku 3.4 se može vidjeti dohvaćanje vrijednosti za napon i struju te izračun snage.

Naime, nakon dohvata u linijama 3 i 10, vrijednosti se postavljaju u pripadne

Ispis 3.4 Dohvaćanje i obrada vrijednosti za napon, struju i snagu unutar funkcije updateBatteryData

```
1
2  //Get voltage
3  val voltageTmp = intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, 0)
4  //Write in global variable
5  voltageV = voltageTmp.toFloat() / 1000 //V
6  //Write in TextView
7  voltageTv.text =  String.format(resources.getString(R.string.voltage_text), voltageV)
8
9  //Get current
10 val currentTmp = getCurrent(this)
11 //Write in global variable
12 currentMA = getCurrent(this).toFloat() // mA - miliAmpers
13 //Write in TextView
14 currentTv.text = String.format(resources.getString(R.string.current_text),currentMA)
15
16 //calculate power and write in global variable
17 powerW = (voltageV  * currentMA / 1000) //Volt * Amp)
18 //Write in TextView
19 powerTv.text =  String.format(resources.getString(R.string.power_text), powerW)
```

globalne varijable u linijama 5 i 12. Istovremeno se izračunava snaga prema jednadžbi 3.1, što se može vidjeti u liniji 17 gdje se vrijednost također zapisuje u globalnu varijablu. Zapisivanje u globalne varijable je potrebno radi pristupa vrijednostima za vrijeme samog testiranja što će se pojasniti u nastavku.

$$Snaga\,(W) = Napon\,(V) * Struja\,(A)\,[8] \tag{3.1}$$

3.1.4 Pokretanje testiranja

Kao što je već spomenuto, testiranje se pokreće klikom na gumb na početnom zaslonu. Prilikom pokretanja testiranja, prvenstveno se izvode tri bitne stvari. Postavlja se okruženje testiranja, resetiraju globalne varijable te pokreće mjerač vremena za odbrojavanje. Namještanje okruženja testiranja predstavlja postavljanje svjetline zaslona i svih zvukova na 50% kao optimalne vrijednosti. Kod resetiranja globalnih varijabli, podrazumijeva se resetiranje varijabli potrebne za spremanje informacija o bateriji kao i varijablu *currentTest* koja služi za spremanje podataka o testu. Brišu

se spremljene iteracije te se postavlja novo ime testa koje se sastoji od korisnikovog unosa i vremenske oznake.

Također, provjerava se da li je upaljen Bluetooth na uređaju kao i povezanost na internet putem mobilne ili WiFi mreže. Pomoću *Toast* razreda, što je, u principu, prikaz koji sadrži kratku poruku za korisnika, ako uređaj ima upaljenu jednu od navedenih komponenata, prikazuje se obavijest za provjeru povezanosti na mrežu i ostale servise.

Kako bi se na učinkovit način provelo testiranje, implementiran je servis *BroadcastTimerService* koji odbrojava vrijeme testiranja. Naime, kako bi mogli izračunati energiju, prema formuli 3.2, potrebno je određeno vrijeme u kojem se razmatra potrošena energija. Stoga, pri početku, odnosno zaustavljanju snimanja, pokreće se ili zaustavlja mjerač vremena za odbrojavanje.

$$Energija\,(J) = Snaga\,(W) * Vrijeme\,(s)\,[8] \tag{3.2}$$

Odbrojavanje vremena unutar servisa *BroadcastTimerService* je implementiran pomoću objekta *CountDownTimer* razreda koji odbrojava od određenog vremena. Stoga, za parametre uzima vrijeme od kojeg mjerač vremena treba odbrojavati te interval za otkucaj u kojem se povratni poziv metode za otkucavanje treba pokrenuti. Naime, *CountDownTimer* sadrži metode za pokretanje odbrojavanja, otkazivanje brojača, zaustavljanje brojača te metodu za otkucavanje intervala koja se poziva pri svakom otkucaju. U ovom slučaju, *CountDownTimer* otkucava svaku sekundu, a implementiran je da uzima proslijeđeno vrijeme za odbrojavanje i interval za novi period analiziranja vrijednosti baterije. Naime, pri odbrojavanju, važno je znati koliko vremena je preostalo do kraja testiranja, kao i kada se postigne željeni interval, to jest, novi period testiranja. Također, računaju se informacije poput trajanja intervala, broja iteracije te sveukupno trajanje testiranja. Prijenos informacija se postiže emitiranjem traženih vrijednosti prilikom svakog otkucaja odnosno prilikom postizanja novog intervala te pri završetku odbrojavanja.

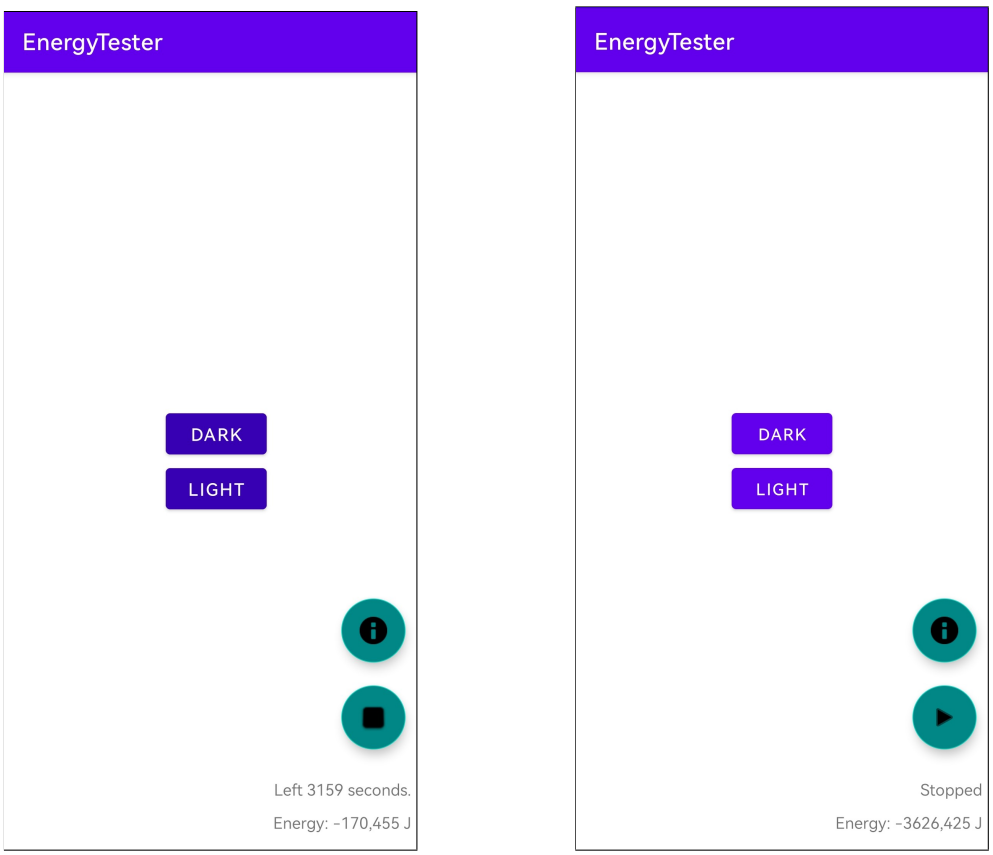
Nakon što je pokrenut brojač te je započeto emitiranje vrijednosti prilikom svakog otkucaja, potrebno je prihvatiti te informacije u *MainActivity* razredu, gdje se odvija ostatak procesa testiranja. Naime, zbog emitiranja potrebnih vrijednosti,

potrebno je postaviti i registrirati prijemnik kao kod primanja vrijednosti baterije. Nakon dohvata emitiranih vrijednosti, na glavnom zaslonu se prikazuje preostalo vrijeme i ukupna potrošena energija do sada. Ako se radi o novom periodu, uzimaju se podaci iz već opisanih globalnih varijabli za energiju, snagu, struju i napon te ujedno uređuju globalne varijable za prosječne vrijednosti istih. Spremaju se podaci o novoj iteraciji, a zatim se ažuriraju vrijednosti trenutačnog testa. Kada pristigne informacija o kraju testa, automatski se prekida snimanje i isključuje servis *BroadcastTimerService*. Također, provedeni test se sprema u listu svih provedenih testova i bazu podataka.

3.2 Postupak

Sam postupak testiranja je jednostavan. Nakon što je izmjenjen kod potreban za testiranje, te je nova verzija aplikacije instalirana na fizički uređaj, najprije je potrebno isključiti sve aktivne procese i pozadinske zadatke. Iako je u aplikaciji implementirano isključivanje mrežnih servisa, potrebno je provjeriti i isključiti sve servise koji mogu utjecati na izvođenje testiranja. Za preciznije rezultate trebala bi se izvaditi SIM kartica, no s obzirom na to da se radi o uređaju koji je svakodnevno u uporabi, taj korak je preskočen.

Nakon pripreme okruženja za izvođenje testa, u aplikaciji je potrebno unijeti parametre poput trajanja testa i intervala te njegovo ime. Naposljetku, pokreće se test te je potrebno ostaviti uređaj na lokaciji gdje će biti na miru do završetka testiranja. Pri početku testiranja, promijeni se svjetlina zaslona i jačina zvuka, a zaslon ostaje upaljen. Također se u donjem desnom kutu zaslona može vidjeti preostalo vrijeme izvršavanja u sekundama te dosadašnju ukupnu potrošenu energiju tijekom testiranja. Kod završetka testiranja, umjesto prikaza sekundi, prikazat će se riječ "*Stopped*". Na slikama 3.3a i 3.3b su prikazane snimke zaslona za vrijeme testiranja i i nakon završetka testiranja svijetle Android teme.



(a) Zaslon tijekom testiranja *(b) Zaslon nakon završenog testiranja*

Slika 3.3 Izgledi zaslona kod testiranja

Poglavlje 4

Testiranje i rezultati

Kako bi se što bolje obuhvatilo cijelo područje razvoja Android aplikacije, testiranja se trebaju provesti na onome što korisnik vidi kao i na dijelu koji se događa u pozadini. Stoga, testiranja se mogu podijeliti i obraditi na dva područja - korisničkom sučelju i programskom dijelu aplikacije. Optimizacija potrošnje energije korisničkog sučelja i programskog koda Android aplikacije ključna je za uspješnu aplikaciju.

Rezultati će biti izraženi pomoću prosječnih vrijednosti struje, napona, snage i ukupne energije za provedeni test u miliamperima, voltima, vatima te džulima. Osim prikaza energije u džulima, kako bi se što bolje predočila potrošnja energije baterije, sagledat će se potrošnja energije kroz sat vremena pomoću vat-sati. U konačnici, za 1 sat, primjenom izraza 2.1, potrošena energija u vat-satima je jednaka potrošenoj snazi. Obzirom da će se u ovom radu potrošnja energije uvijek razmatrati unutar sat vremena, energija u vat-satima i snaga se mogu izjednačiti. Time se, u ovom radu, s džulima izračunava potrošena energija u određenom vremenskom periodu, dok se s vatima dobiva informacija o potrošnji snage odnosno energije u sat vremena. Sve vrijednosti u tablicama će biti zaokružene na 5 decimala.

$$Kapacitet\,energije\,(Wh) = Napon\,(V) * Struja\,(A) * Vrijeme\,(h)\,[9] \qquad (4.1)$$

Kako bi rezultate smatrali pouzdanima, nakon svake analize podataka, ovisno o broju slučaja, provest će se t-test odnosno ANOVA test s ponavljajućim mjerenjima

i post hoc analizom. Prvenstveno se testira sljedeća nulta hipoteza:

H0: nema razlike između energetske učinkovitosti implementacija koda koje se smatraju neučinkovitim i onih koje se smatraju učinkovitima

4.1 Korisničko sučelje

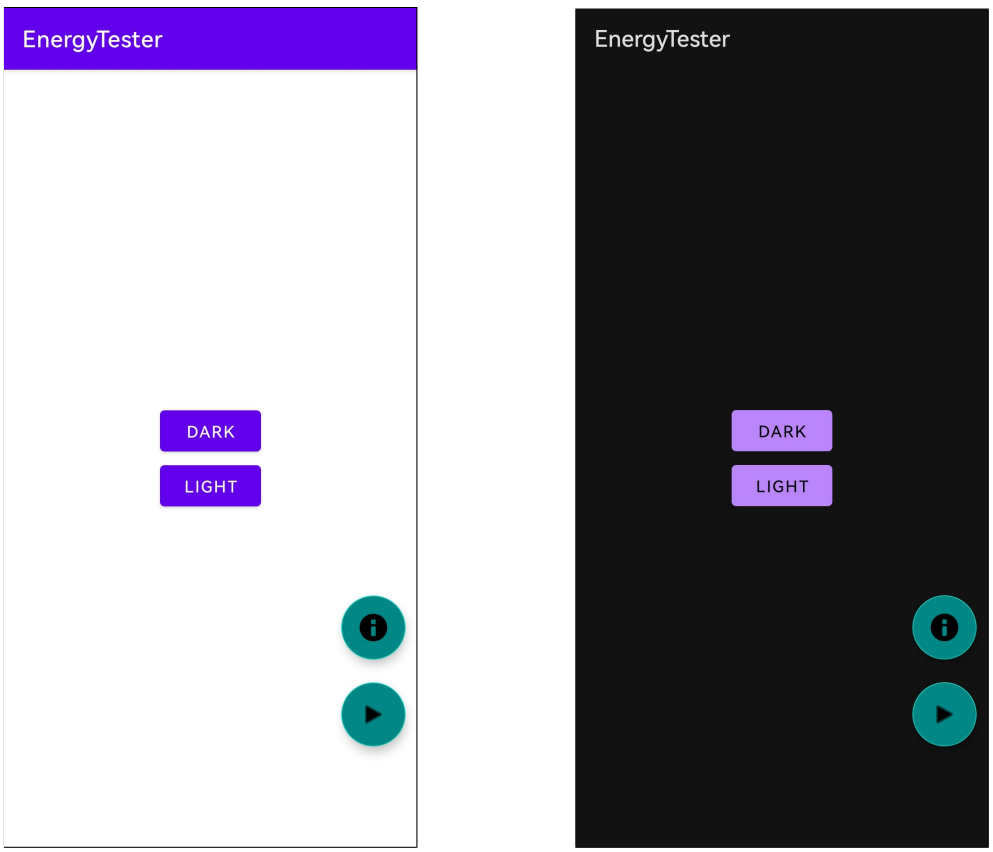
Potrošnja energija korisničkog sučelja postaje sve važniji čimbenik koji treba uzeti pri razvoju mobilnih aplikacija. Grafičko korisničko sučelje se može smatrati kao energetski intenzivniji dio Android aplikacije s obzirom na to da zahtijeva puno procesorske snage za mjerenje i iscrtavanje elemenata. Kako korisnici postaju svjesniji trajanja baterije svojih uređaja, pogotovo pri aktivnom korištenju aplikacije gdje veliku ulogu ima korisničko sučelje, programeri moraju osigurati da njihove aplikacije poduzimaju korake za smanjenje potrošnje energije. To može uključivati optimiziranje elemenata korisničkog sučelja za minimalno trošenje energije, kao što je korištenje određenih boja te odabir odgovarajućih rasporeda i atributa elemenata [10]. Uzimajući u obzir dobivene rezultate opisane u idućim potpoglavljima, razvojni programeri mogu pružiti da njihove aplikacije ne troše prebrzo bateriju uređaja korisnika, a istovremeno pružaju privlačno korisničko iskustvo.

4.1.1 Teme i boje

Kao jedan od bitnijih dijelova za ugodnu interakciju korisnika s aplikacijom je odabir boja u korisničkom sučelju. Boje s visokim kontrastom mogu smanjiti količinu energije koju troše zasloni telefona jer zahtijevaju manje energije za prikaz [11]. Osim toga, tamne boje mogu smanjiti količinu svjetla koje emitira zaslon, čime se smanjuje količina potrošene energije. Odabir pravih boja za korisničko sučelje stoga može imati značajan utjecaj na potrošnju energije aplikacije.

U ovome potpoglavlju će se obraditi testiranje boja na principu tamne i svjetle Android teme. Naime, tamna tema koristi tamnije boje kao što je crna boja dok svjetla tema koristi svjetlije boje, primjerice bijelu boju. Shodno tome, u ovom slučaju, pozadina aplikacije se mijenja ovisno o temi pomoću dva gumba. Izgled koji

će se testirati je prikazan na slikama 4.1a i 4.1b u svjetlijoj i tamnoj verziji s dva gumba čijim pritiskom se mijenja tema i ime testa.



(a) Svjetla Android tema *(b) Tamna Android tema*

Slika 4.1 Izgled zaslona kod testiranja boja pomoću Android tema

Implementacija promjene teme je prikazana na isječku 4.1 gdje su realizirani osluškivači klika za pojedini gumb.

Ispis 4.1 Programski dio za testiranje boja pomoću Android tema

```
findViewById<Button>(R.id.btn_changeTheme_dark).setOnClickListener {
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES)
    testName = "test_themes_dark"
}
findViewById<Button>(R.id.btn_changeTheme_light).setOnClickListener {
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)
    testName = "test_themes_light"}
```

Nakon odabira teme, definiraju se ostali parametri potrebni za testiranje. Vrijeme testiranja je postavljeno na 60 minuta, a interval uzimanja podataka na 30 sekundi. Test se ponovio 10 puta za svaku temu što rezultira ukupnim zbrojem od 20 testova.

Uzevši u obzir sve testove, dolazi se do prosječnih vrijednosti struje, napona i snage. Također se računa i prosječna ukupna energija izvedenih testova. Stoga, rezultatima prikazanima tablicom 4.1 može se uočiti da je tamna tema potrošila manje energije nego svjetla i to za 192,79 J. Također, tamna tema troši manje snage od 0,94 W za razliku od svjetle koja troši približno 1 W.

Tablica 4.1 Rezultati testiranja tamne i svjetle teme izraženi prosjekom od 10 testiranja po 60 minuta

Tema	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
Svjetla	258,85915	3,85728	0,99807	3592,90549	17,69644
Tamna	244,41588	3,86439	0,94448	3400,11877	83,25831

Istodobno, ušteda energije se može gledati iz perspektive vremenskog trajanja baterija. Kao što je već spomenuto, uređaj ima kapacitet baterije od 15,96 Wh. Razmatrajući izračunatu snagu koja se troši u sat vremena, može se reći da, pri potpunom kapacitetu od 15,96 Wh, koristeći svjetlu temu, baterija će se isprazniti za otprilike 16 sati, dok koristeći tamnu temu za otprilike 17 sati. Zaključno, primjenjujući tamnu temu se može uštedjeti do sat vremena rada baterije.

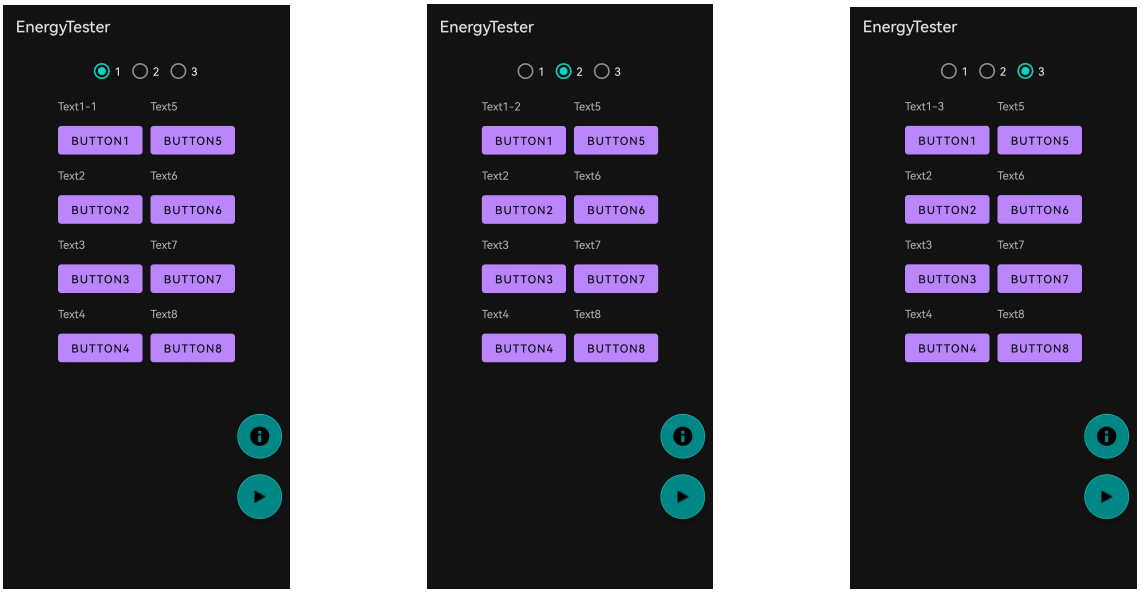
Za kraj, potrošnja energije za svjetlu i tamnu temu su analizirane pomoću t-testa. Dobivenim rezultatima je utvrđeno da je potrošnja energije za svjetlu temu ($3592{,}91 \pm 83{,}26$ J) statistički značajno veća od potrošnje za tamnu temu ($3400{,}12 \pm 17{,}70$ J): $t(9) = 6{,}22$; $p < 0{,}001$.

Analizom se mogu prihvatiti dobiveni zaključci i zasigurno potvrditi da korištenje tamnijih boja i tamne Android teme može uštedjeti energiju zbog manje količine svjetlosti koje zaslon emitira.

4.1.2 Ugniježđivanje rasporeda

Ugniježđivanje elemenata i rasporeda u Android aplikaciji može utjecati na performanse na više načina. Budući da se svaki ugniježđeni element mora zasebno renderirati i upravljati njime, to može dovesti do smanjenja ukupne izvedbe, osobito ako postoji mnogo ugniježđenih elemenata u hijerarhiji [12]. Osim toga, ugniježđivanje elemenata može pridonijeti povećanom korištenju memorije, budući da svaki element treba pohraniti vlastito stanje i podatke. Konačno, elementi za ugniježđivanje mogu povećati složenost korisničkog sučelja, otežavajući otklanjanje pogrešaka i optimizaciju problema s izvedbom.

Obzirom da se u ovom radu stavlja fokus na potrošnju energije, postavlja se pitanje koliko ugniježđivanje rasporeda i velika hijerarhija elemenata igra ulogu kod energetske potrošnje aplikacije. Kako bi se dobio što bolji uvid, testiraju se tri različite hijerarhije rasporeda. Sva tri rasporeda prikazuju iste elemente u istom poretku što se može vidjeti na slikama 4.2, ali njihova implementacija je različita kao što je prikazano slikama 4.3 gdje su prikazane hijerarhije rasporeda pomoću *LayoutInspector* alata sadržanim u Android Studiju.



(a) Raspored 1 *(b) Raspored 2* *(c) Raspored 3*

Slika 4.2 Izgled zaslona od tri rasporeda za testiranje hijerarhije i ugniježđivanja

Kako bi se zasigurno i lakše promijenio raspored za testiranje, implamentiran je radio gumb čijim klikom se prikazuje određeni raspored, a na svakom izgledu je uz tekst *Text1* postavljen broj pripadajućeg rasporeda.

(a) Raspored 1 *(b) Raspored 2* *(c) Raspored 3*

Slika 4.3 Hijerarhija za tri rasporeda prikazana pomoću LayoutInspectora

Prvi raspored, prikazan slikama 4.2a i 4.3a, se sastoji od nepotrebnih ugniježđivanja elemenata. Koristeći *LinearLayout* raspored, dobiva se što veća hijerarhija s dubinom od 5 ugniježđenih elemenata.

Drugi raspored, prikazan slikama 4.2b i 4.3b, sastoji se od *LinearLayout* rasporeda, isto kao i prethodni raspored ali na optimalan način sa što manje elemenata. Nastojeći smanjiti hijerarhiju, dobiva se dubina od 2 ugniježđena elementa.

Treći raspored, prikazan slikama 4.2c i 4.3c, predstavlja najbolju implementaciju traženog izgleda. Sastoji se od jednog *GridLayout* elementa te time ima najmanju dubinu od 1 ugniježđenog elementa.

Programski dio testa, prikazan na isječku 4.2, sastoji se od promjene prikazanog rasporeda pomoću tri radio gumba. Također se mijenja ime testa ovisno o testiranom rasporedu. Kako bi se testirala potrošnja energije koju iscrtavanje rasporeda troši, postavlja se neprestano izvršavanje iscrtavanje rasporeda svake milisekunde pomoću svojstva vidljivosti.

Test se proveo za sva tri različita hijerarhijska rasporeda u trajanju od 10 minuta. Za svaki raspored, test se ponovio 30 puta što rezultira od 90 ukupno napravljenih testova. Interval uzimanja vrijednosti baterije bio je postavljen na 1 sekundu.

Kao rezultat, dobivene vrijednosti za prosječnu struju, napon, snagu i ukupnu energiju su prikazani tablicom 4.2. Kao što je očekivano, raspored 1 koji ima najveću dubinu je potrošio najviše energije od 926,09 J. Zatim slijedi raspored 2 s 898,06 J te raspored 3 s 891,01 J. Gledajući sa aspekta potrebne snage, raspored 1 ima najveću snagu od 1,54 W, dok raspored 3 najmanju od 1,49 W. Iako postoji samo mala energetska razlika od 7 J između rasporeda 2 i 3, može se pretpostaviti da raspored 3 ima najmanju potrošnju energije, što je u skladu s očekivanjima. U usporedbi s rasporedom 1, raspored 3 je ostvario uštedu od 35,08 J energije.

Rezultati se dodatno mogu razmotriti u intervalu od 60 minuta pomoću potrošene snage te usporediti kroz vremensko trajanje baterije. Prema tim vrijednostima, kod potpune napunjenosti baterije od 15,96 Wh, raspored 1 će iscrpiti bateriju za 10 sati i 20 minuta, raspored 2 za 10 sati i 39 minuta, a raspored 3 za 10 sati i 44 minute, što daje apsolutnu razliku od 24 minute.

Ispis 4.2 Programski dio za testiranje hijerarhije rasporeda

```
1
2  val layout_container : RelativeLayout = findViewById(R.id.layout_inflater)
3  var inflateLayout = layoutInflater.inflate(R.layout.layout_hierarchy_layout1, null)
4  layout_container.addView(inflateLayout)
5  testName = "test_LH_l1"
6  findViewById<EditText>(R.id.test_name).setText(testName)
7  findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
8  RadioGroup.OnCheckedChangeListener { group, checkedId ->
9      inflateLayout = when (checkedId) {
10         R.id.layout1 -> layoutInflater.inflate(
11                         R.layout.layout_hierarchy_layout1, null)
12         R.id.layout2 -> layoutInflater.inflate(
13                         R.layout.layout_hierarchy_layout2, null)
14         else -> layoutInflater.inflate(R.layout.layout_hierarchy_layout3, null)
15     }
16     testName = when (checkedId) {
17         R.id.layout1 -> "test_LH_l1"
18         R.id.layout2 -> "test_LH_l2"
19         else -> "test_LH_l3"
20     }
21     findViewById<EditText>(R.id.test_name).setText(testName)
22     layout_container.removeAllViews()
23     layout_container.addView(inflateLayout)
24     layout_container.visibility = View.VISIBLE
25 })
26
27 fixedRateTimer("timer", false, 0, 1) {
28     this@MainActivity.runOnUiThread {
29         if (recording) {
30             layout_container.visibility = View.GONE
31             layout_container.visibility = View.VISIBLE
32         }
33     }
34 }
```

Iako dobiveni rezultati ukazuju da je raspored 3 energetski najučinkovitiji, potrebno je provesti ANOVA RM test. Njime je utvrđeno da postoji značajna razlika u potrošnji energije ovisno o hijerarhiji izgleda: $F(2,58) = 32{,}223$; $p < 0{,}001$. Post-hoc analiza, koja uključuje međusobne usporedbe s Bonferronijevom korekcijom, utvrdila je sljedeće:

- Potrošnja energije kod korištenja rasporeda 1 ($926{,}09 \pm 22{,}58$ J) je statistički značajno veća od potrošnje energije kod korištenja rasporeda 2 ($898{,}06 \pm 20{,}83$ J): $p < 0{,}001$.

- Potrošnja energije kod korištenja rasporeda 1 ($926{,}09 \pm 22{,}58$ J) je statistički značajno veća od potrošnje energije kod korištenja rasporeda 3 ($891{,}01 \pm 23{,}74$ J): $p < 0{,}001$.
- Razlika u potrošnji energije kod korištenja rasporeda 2 ($898{,}06 \pm 20{,}83$ J) i kod korištenja rasporeda 3 ($891{,}01 \pm 23{,}74$ J) nije statistički značajna: $p = 0{,}051 > 0{,}05$.

Iako je postavljeno konstantno izvršavanje iscrtavanja elemenata što se u stvarnosti ne događa, ipak se može zaključiti da korištenjem rasporeda 2 ili 3 umjesto rasporeda 1, može uštediti energija baterije. Ovo jasno ukazuje na činjenicu da hijerarhija rasporeda i ugniježđivanje elemenata imaju utjecaj na potrošnju energije uređaja.

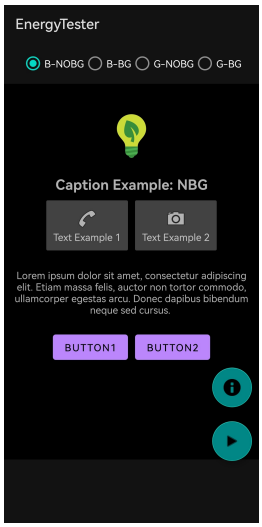
Tablica 4.2 Rezultati testiranja 3 različita rasporeda elemenata izraženi prosjekom od 30 testiranja po 10 minuta

Raspored	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
1.	391,43542	3,94357	1,54348	926,08854	22,58091
2.	378,72603	3,95283	1,49677	898,05610	20,83155
3.	377,10173	3,93793	1,48499	891,00574	23,74253

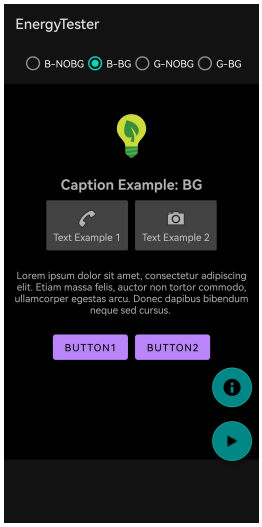
4.1.3 Precrtavanje

Svaki element korisničkog sučelja može imati postavljenu pozadinu. Međutim, kada se više elemenata preklapa na zaslonu i svi imaju postavljenu pozadinu, dolazi do nepotrebnog precrtavanja piksela na zaslonu [13]. Naime, svaki element se pojedinačno procesira i prikazuje, što zahtijeva procesorsku snagu. Stoga, kod precrtavanja dolazi do bespotrebnog gubitka vremena za prikazivanje piksela koji će kasnije biti prekriveni drugim elementima. Ako se isti pikseli nepotrebno iscrtavaju više puta, to može utjecati na izvedbu korisničkog sučelja aplikacije te ujedno povećati potrošnju energije uređaja .

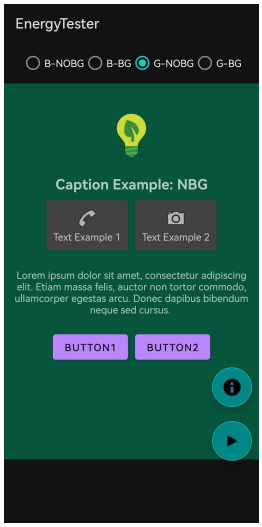
Cilj je testirati potrošenu snagu na iscrtavanje jedne ili više pozadina. U tu svrhu, testirat će se dva izgleda koji izgledaju identično na zaslonu, ali se razlikuju u implementaciji. Dodatno, test će se provesti u dvije boje - crnoj i zelenoj. Na slikama 4.4 se može vidjeti prikaz zaslona za test precrtavanja pozadina pomoću crne i zelene boje. Konkretno, dva izgleda će uključivati jednu pozadinu, dok će druga dva izgleda imati više pozadina, točnije 6 pozadina, svaki u crnoj i zelenoj boji.



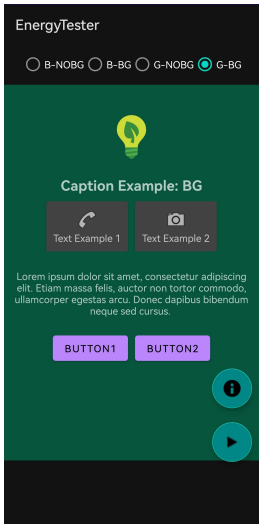
(a) Jedna crna pozadina



(b) Više crnih pozadina



(c) Jedna zelena pozadina



(d) Više zelenih pozadina

Slika 4.4 Izgled zaslona kod testiranja precrtavanja pozadina

U nastavku je prikazan isječak 4.3 koji sadrži programski kod za mjenjanje prikazanog rasporeda te neprestano iscrtavanje istog.

Ispis 4.3 Programski dio za testiranje precrtavanje pozadina rasporeda

```
1
2  val layout_container : RelativeLayout = findViewById(R.id.layout_inflater)
3  var inflateLayout = layoutInflater.inflate(
4                      R.layout.layout_background_overdraw_black_nobg, null)
5  layout_container.addView(inflateLayout)
6  testName = "test_background_overdraw_blackNOBG"
7  findViewById<EditText>(R.id.test_name).setText(testName)
8  findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
9  RadioGroup.OnCheckedChangeListener { group, checkedId ->
10     inflateLayout = when (checkedId) {
11         R.id.layout1 -> layoutInflater.inflate(
12                         R.layout.layout_background_overdraw_black_nobg, null)
13         R.id.layout2 -> layoutInflater.inflate(
14                         R.layout.layout_background_overdraw_black_bg, null)
15         R.id.layout3 -> layoutInflater.inflate(
16                         R.layout.layout_background_overdraw_green_nobg, null)
17         else -> layoutInflater.inflate(
18                 R.layout.layout_background_overdraw_green_bg, null)
19     }
20     testName = when (checkedId) {
21         R.id.layout1 -> "test_background_overdraw_blackNOBG"
22         R.id.layout2 -> "test_background_overdraw_blackBG"
23         R.id.layout3 -> "test_background_overdraw_greenNOGB"
24         else -> "test_background_overdraw_greenBG"
25     }
26     findViewById<EditText>(R.id.test_name).setText(testName)
27
28     layout_container.removeAllViews()
29     layout_container.addView(inflateLayout)
30     layout_container.visibility = View.VISIBLE
31 })
32
33 fixedRateTimer("timer", false, 0, 1) {
34     this@MainActivity.runOnUiThread {
35         if (recording) {
36             layout_container.visibility = View.GONE
37             layout_container.visibility = View.VISIBLE
38         }
39     }
40 }
```

Test je trajao 10 minuta za svaki raspored. Testiranje se ponovilo 30 puta za pojedini raspored što daje sveukupno 120 napravljenih testova za precrtavanje pozadina.

Tokom testiranja, vrijednosti baterije su uzimane svaku sekundu.

Nakon testiranja, dobivene su vrijednosti prikazane tablicom 4.3 za crnu pozadinu te tablicom 4.4 za zelenu pozadinu. Iskazane su prosječne vrijednosti struje, napona, snage i ukupne energije. Očekivalo se da će raspored s jednom pozadinom u crnoj boji trošiti najmanje energije, zbog manjeg broja precrtavanja i korištenja tamnije boje, što je potvrđeno prethodnim testom kao energetski učinkovitije rješenje.

Tablica 4.3 Rezultati testiranja precrtavanje pozadine elemenata pomoću crne pozadine izraženi prosjekom od 30 testiranja po 10 minuta

Broj pozadina	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
Jedna pozadina	309,97532	3,96054	1,22747	736,48647	20,47231
Više pozadina	313,25111	3,96312	1,24150	744,89389	22,23234

Tablica 4.4 Rezultati testiranja precrtavanje pozadine elemenata pomoću zelene pozadine izraženi prosjekom od 30 testiranja po 10 minuta

Broj pozadina	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$[mA]$	$[V]$	$[W]$	$[J]$	$[Wh]$
Jedna pozadina.	317,41262	3,95606	1,25555	753,32504	13,16377
Više pozadina.	319,28961	3,97728	1,26984	761,90745	13,59879

Pomoću dobivenih vrijednosti, može se pretpostaviti da precrtavanje piksela pridonosi većoj potrošnji energije. Naime, s rezultatom od prosječno potrošenih 736,49 J u 10 minuta, raspored s jednom crnom pozadinom je potrošio najmanje energije, dok je raspored s više zelenih pozadina potrošio najviše i to 761,91 J. Iako se ta dva rasporeda, osim po broju pozadina, razlikuju u korištenoj boji, svakako se može pretpostaviti da precrtavanje troši više resursa. Budući da se kod korištenja crne pozadine rezultati razlikuju u otprilike 8,41 J, dok se korištenjem zelene pozadine energija razlikuje u 8,58 J, može se očekivati da korištenje više pozadina u rasporedu utječe na energiju.

Međutim, potrošnja energije se može analizirati koristeći potrošenu snagu unutar sat vremena. Naime, vrijednosti potvrđuju dosadašnje tvrdnje gdje, iako je mala razlika, gledajući iz perspektive vremenskog trajanja baterije, kod upotrebe crne pozadine, baterija će se potrošiti za 13 sati s jednom pozadinom, a s više pozadina za 12 sati i 51 minutu. Također kod upotrebe zelene boje, baterija će se potrošiti za 12 sati i 42 minute s jednom pozadinom, dok s više pozadina će trebati 12 sati i 34 minute.

Kako bi potvrdili dobivene tvrdnje, provodi se t-test za oba slučaja, s crnom i zelenom pozadinom. U slučaju crne pozadine, analizom je utvrđeno da razlika u potrošnji energije između izgleda s jednom pozadinom ($736{,}49 \pm 20{,}47\,\text{J}$) i izgleda s više pozadina ($744{,}89 \pm 22{,}23\,\text{J}$) nije statistički značajna: $t(29) = 1{,}720$; $p = 0{,}096 > 0{,}05$. Kod korištenja zelene pozadine, utvrđeno je da nema statistički značajne razlike između potrošnje energije kod korištenja izgleda s jednom pozadinom ($753{,}330 \pm 0{,}136\,\text{J}$) i korištenja izgleda s više pozadina ($761{,}91 \pm 13{,}16\,\text{J}$): $t(29) = 1{,}795$; $p = 0{,}083 > 0{,}05$.

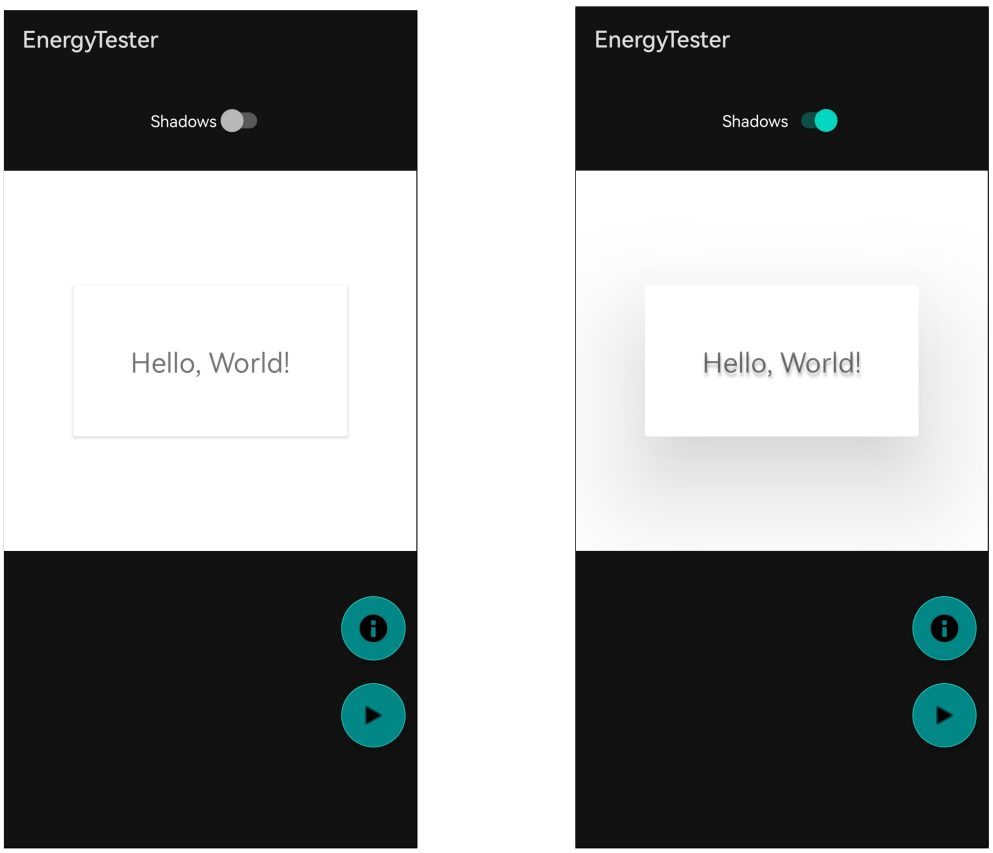
Obzirom da razlika u potrošenoj energiji u niti jednom slučaju nije bila značajna, ne može se sa sigurnošću potvrditi da precrtavanje pozadina povećava potrošnju energije. Za postizanje veće statističke značajnosti rezultata u budućnosti, pretpostavlja se da bi povećanje broja pozadina moglo rezultirati većim brojem ponovno iscrtanih piksela. U takvom scenariju, moguća je značajna razlika u potrošnji energije.

4.1.4 Sjene

Često se zbog vizualnog efekta dodaju sjene na elemente. Njima se pruža trodimenzionalan izgled čime se poboljšava estetski dojam i korisničko iskustvo. Sjene se mogu primijeniti na elemente kao što su kartice (*CardView*) i tekstualna polja (*TextView*) kako bi se istaknula njihova dubina i vizualna hijerarhija u odnosu na druge elemente na zaslonu. Međutim, to je dodatan zahtjev za grafičko računanje i iscrtavanje piksela što može negativno utjecati na energetsku potrošnju uređaja [10].

Zbog boljeg uvida u energetsku potrošnju, biti će izvršeno testiranje korištenja sjena. Obzirom da se sjena najčešće primjenjuje na elementima kartice, isti je korišten u testiranju. Dodatno, unutar kartice je postavljen tekst koji također ima pridodanu

sjenu. Metodologija testiranja će se temeljiti na održavanju istog izgleda za oba slučaja, s razlikom da će u jednom slučaju sjene biti isključene, odnosno uklonjene, dok će u drugom slučaju biti uključene, odnosno postavljene. Na slikama 4.5a i 4.5b se mogu vidjeti snimke zaslona nad kojima se provodi testiranje.



(a) Uklonjene sjene *(b) Postavljene sjene*

Slika 4.5 Izgled zaslona kod testiranja sjena

Također, na programskom isječku 4.4 je prikazan programski kod kojim se uključuju i isključuju sjene pomoću preklopnog gumba. Tokom testiranja, neprekidno svake milisekunde, iscrtava se element kartice, a ujedno i sadržan tekst, pomoću atributa vidljivosti.

Izvršeno je ukupno 60 testova, pri čemu je svaki slučaj testiran 30 puta u trajanju od 10 minuta. Interval uzimanja vrijednosti baterije bio je postavljen na 1 sekundu.

Ispis 4.4 Programski dio za testiranje sjena

```
1
2  val cardView = findViewById<CardView>(R.id.cardView_shadow)
3  val textView = findViewById<TextView>(R.id.textView_shadow)
4  testName = "test_shadows_off"
5  findViewById<EditText>(R.id.test_name).setText(testName)
6  findViewById<SwitchCompat>(R.id.switch_shadow).setOnCheckedChangeListener{
7          _, isChecked ->
8      if(!isChecked) {
9          cardView.elevation = 2f
10         cardView.translationZ = 0f
11         textView.setShadowLayer(0f,0f,0f,Color.TRANSPARENT)
12         testName = "test_shadows_off"
13     } else {
14         cardView.elevation = 10f
15         cardView.translationZ = 200f
16         textView.setShadowLayer(6f,2f,10f,Color.GRAY)
17         testName = "test_shadows_on"
18     }
19
20     findViewById<EditText>(R.id.test_name).setText(testName)
21
22 }
23
24 fixedRateTimer("timer", false, 0, 1) {
25     this@MainActivity.runOnUiThread {
26         if (recording) {
27             cardView.visibility = View.GONE
28             cardView.visibility = View.VISIBLE
29         }
30     }
31 }
```

Nakon svih izvršenih testova, dobiveni su rezultati prikazani tablicom 4.5. Razmatrajući prosječne potrošnje energije za konstanto iscrtavanje elemenata sa i bez sjena, kao što je očekivano, korištenje elementa s postavljenim sjenama zahtijeva više energije i to za 22,95 J.

S aspekta vremenskog trajanja baterije, u ovom slučaju pri potpunom kapacitetu energije od 15,96 Wh, elementi bez sjena će uštediti otprilike pola sata trajanja baterije do potpunog pražnjenja. Naime, obzirom na snagu u iznosu od 1,24 i 1,17, korištenje elemenata sa sjenama će isprazniti bateriju za 13 sati i 38 minuta, dok će iscrtavanje elemenata bez sjena iskoristiti bateriju kroz 14 sati i 6 minuta.

Tablica 4.5 Rezultati testiranja korištenja sjena na elementima izraženi prosjekom od 30 testiranja po 10 minuta

Sjene	**Struja** $\overline{I}\,[mA]$	**Napon** $\overline{U}\,[V]$	**Snaga** $\overline{P}\,[W]$	**Ukupna energija** $\overline{E}\,[J]$	σ
Postavljene	313,25111	3,96312	1,24150	701,95363	7,37027
Uklonjene	296,52928	3,94573	1,16996	679,00632	10,99620

Dodatno, t-testom je ustanovljeno da je potrošnja energije koristeći postavljene sjene ($701{,}95 \pm 7{,}37$ J) statistički značajno veća nego potrošnja energije s uklonjenim sjenama ($679{,}01 \pm 10{,}10$ J): $t(29) = 9{,}215$; $p < 0{,}001$.

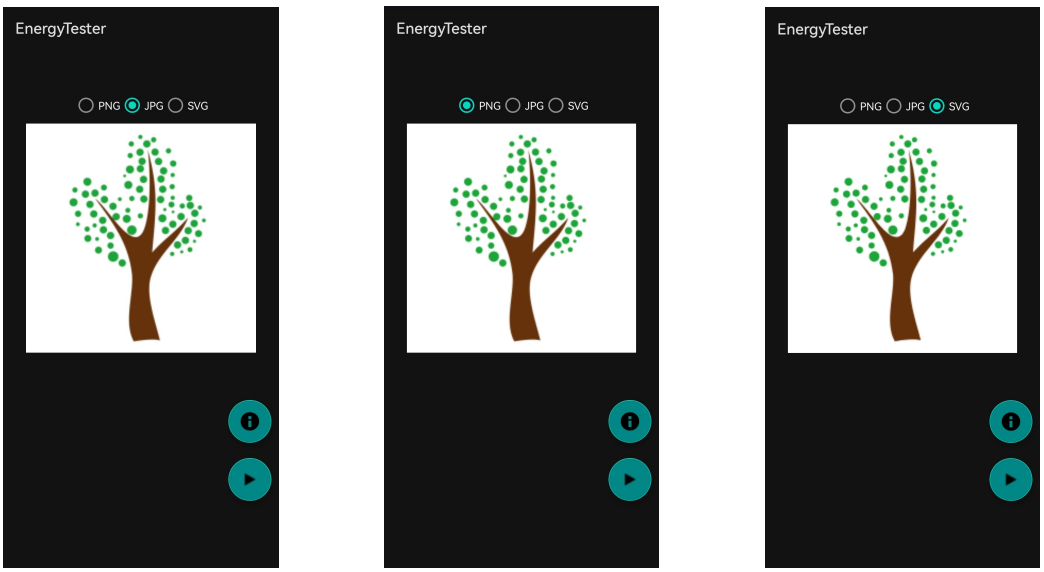
Zaključno, može se potvrditi da korištenje sjena iziskuje više energije. Iako sjene pridodaju poseban faktor korisničkom sučelju, važno je znati da njihova upotreba može povećati potrošnju energije. Stoga je ključno prilagoditi implementaciju sjena na način da njihova upotreba bude količinski umjerena.

4.1.5 Format slike

U aplikacijama se često koriste slike te se koriste razni formati ovisno o njihovim karakteristikama. Različiti formati imaju različite metode kompresije i karakteristike koje mogu utjecati na veličinu datoteke i resurse potrebne za prikazivanje slike [14]. Važno je prilagoditi format slika prema potrebama aplikacije, no u situacijama gdje format slike nije presudan, moguće je koristiti format koji ima manje energetske zahtjeve

Za bolji budući odabir formata slika, provest će se testiranje postavljanje slike u različitim formatima. Za testiranje su uzeti formati JPG (eng. *Joint Photographic Experts Group*), PNG (eng. *Portable Network Graphics*) i SVG (eng. *Scalable Vector Graphics*) kao popularniji formati slika. Svaki od ovih formata ima svoje karakteristike i prednosti. JPG format je široko prihvaćen i koristi se za komprimiranje slika visoke kvalitete, dok PNG format omogućuje bolju transparentnost i podržava slike s prozirnim pozadinama. S druge strane, SVG format je vektorski format koji omogućuje skaliranje slika bez gubitka kvalitete. U ovom slučaju, testira se postav-

ljanje slike, a element za sliku, prikazan na snimkama zaslona 4.6, u sva tri slučaja je identičan.



(a) JPG *(b) PNG* *(c) SVG*

Slika 4.6 Izgled zaslona kod testiranja formata slike

Na programskom isječku 4.5 je prikazan programski kod za testiranje postavljanja različitih foramta slike. Njime se mijenja korištenje formata te se ujedno postavlja izvor slike na zadani format. Dodatno, tokom testiranja, element slike neprekidno mijenja izvor, odnosno izvor slike se poništava i ponovno stavlja na zadani format.

Za svaki format se provelo 20 testiranja po 10 minuta s intervalom uzimanja podataka baterije od 1 sekunde. Time se dobiva ukupno 60 provedenih testova.

Nakon svih testova, izražene su prosječne vrijednosti za struju, napon, snagu i ukupnu energiju tablicom 4.6. Prema tim vrijednostima, formati JPG i PNG imaju poprilično jednak energetski učinak, dok se potrošnja energije koristeći SVG format povećala, i to za otprilike 42 J. Također, gledajući snagu, SVG format koristi 0,07 W više od ostalih formata.

Nadalje, može se promatrati trajanje baterije uređaja. Naime, PNG i JPG format, s približno jednakom potrošnjom od 1,24 W, ispraznit će bateriju s punim kapacitetom od 15.96 Wh u otprilike istom vremenskom razdoblju. Drugim rije-

Ispis 4.5 Programski dio za testiranje formata slike

```
val image = findViewById<ImageView>(R.id.image_format)
testName = "test_image_format_png"
findViewById<EditText>(R.id.test_name).setText(testName)
var imageResource = R.mipmap.ic_image_png_format
findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
    RadioGroup.OnCheckedChangeListener { group, checkedId ->
        imageResource = when (checkedId) {
            R.id.format1 -> R.mipmap.ic_image_png_format
            R.id.format2 -> R.mipmap.ic_image_jpg_format
            else -> R.mipmap.ic_image_svg_format
        }
        image.setImageResource(imageResource)
        testName = when (checkedId) {
            R.id.format1 -> "test_image_format_png"
            R.id.format2 -> "test_image_format_jpg"
            else -> "test_image_format_svg"
        }
        findViewById<EditText>(R.id.test_name).setText(testName)
    })
fixedRateTimer("timer", false, 0, 1) {
    this@MainActivity.runOnUiThread {
        if (recording) {
            image.setImageResource(0)
            image.setImageResource(imageResource)
        }
    }
}
```

čima, JPG format će iscrpiti bateriju za 12 sati i 52 minute, a PNG za 12 sati i 54 minute. S druge strane, koristeći SVG format s potrošnjom od 1,31 Wh, baterija će trajati 12 sati i 11 minuta. U ovom slučaju, to rezultira uštedom od 42 minute rada baterije koristeći JPG ili PNG format.

Dodatno, provedena je analiza ANOVA RM testom. Time je ustanovljeno da postoji značajna razlika u potrošnji energije ovisno o korištenom formatu slike: $F(2,38) = 94{,}819$; $p < 0{,}001$. Post-hoc analizom, koja uključuje međusobne usporedbe primjenom Bonferronijevom korekcijom, utvrđeno je sljedeće:

- Razlika u potrošnji energije kod korištenja formata JPG ($743{,}95 \pm 10{,}62\,\mathrm{J}$) i korištenja formata PNG ($742{,}09 \pm 10{,}60\,\mathrm{J}$) nije statistički značajna: $p = 0{,}06 > 0{,}05$.

Tablica 4.6 Rezultati testiranja postavljanja izvora slike u različitim formatima izraženi prosjekom od 20 testiranja po 10 minuta

Format	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
PNG	313,75834	3,94216	1,23685	742,087054	10,60200
JPG	313,31457	3,95768	1,23994	743,94993	10,62030
SVG	330,92581	3,95570	1,30896	785,26517	12,03497

- Potrošnja energije kod korištenja formata JPG ($743{,}95 \pm 10{,}62$ J) je statistički značajno manja od potrošnje energije kod korištenja formata SVG ($785{,}27 \pm 12{,}03$ J): $p < 0{,}001$.
- Potrošnja energije kod korištenja formata PNG ($742{,}09 \pm 10{,}60$ J) je statistički značajno manja od potrošnje energije kod korištenja formata SVG ($785{,}27 \pm 12{,}03$ J): $p < 0{,}001$.

Iako se porast energije temelji na konstantom postavljanju izvora slike, zasigurno se može zaključiti da postavljanje SVG formata zahtijeva više energije nego JPG ili PNG formata.

4.2 Programski kod

Programski kod igra ključnu ulogu u izvođenju Android aplikacija i postizanju njihove učinkovitosti. Kvalitetan programski kod karakterizira optimizacija, jednostavnost i strukturiranost. Prilikom razvoja aplikacije, programeri posvećuju pažnju optimizaciji programskog koda kako bi osigurali brzo izvršavanje operacija uz efikasno korištenje resursa. Međutim, kada je riječ o energetskoj učinkovitosti, aspekti kao što su vrijeme izvršavanja operacija ili rukovanje memorijom i drugim resursima nisu jedini faktori koji treba uzeti u obzir. Važno je također minimizirati potrošnju energije tijekom izvođenja aplikacije. To uključuje smanjenje upotrebe energetski zahtjevnih operacija i podatkovnih struktura. Kroz sveobuhvatan pristup koji kombinira optimizaciju izvedbe i smanjenje potrošnje energije, programeri mogu stvoriti

energetski učinkovite Android aplikacije koje pružaju visoku izvedbu uz minimalan utjecaj na trajanje baterije uređaja.

4.2.1 Podatkovne strukture

Za pravilnu i učinkovitu izvedbu aplikacije, jedan od ključnih izbora je odabir prikladne podatkovne strukture za određeni scenarij. Važno je pažljivo razmotriti izbor podatkovnih struktura jer one mogu imati različitu učinkovitost u smislu brzine izvršavanja i potrošnje resursa, uključujući energiju [15]. Prilagođavanje podatkovnih struktura prema potrebama aplikacije može rezultirati optimizacijom energetske potrošnje, poboljšanjem performansi i dugotrajnosti baterije na Android uređajima.

Obzirom na široki izbor podatkovnih struktura u Android razvoju, izbor se sveo na testiranje strutkura podataka ArrayMap i HashMap. Naime, obje strukture su kolekcije elemenata po principu ključ-vrijednost. No, kod HashMap strukture, ključevi se označavaju i pohranjuju u veliko polje, što može rezultirati neučinkovitošću memorije. S druge strane, ArrayMap struktura je memorijom učinkovitija alternativa, koristeći dva manja polja za pohranu ključeva i vrijednosti [16]. Međutim, pristupanje vrijednostima u strukturi ArrayMap može biti sporije s povećanjem broja elemenata. Stoga se za male skupove podataka preporuča upotreba ArrayMap strukture, dok je HashMap općenito poželjniji za ostale slučajeve. To će se testirati i potvrditi ovim radom.

Podatkovne strukture će se testirati u 2 slučaja - s malim i velikim skupom elemenata. Za mali skup će se uzeti 200 elemenata, a za veliki skup 2000 elemenata. Za svaku strukturu podataka će se provesti iste operacije koje uključuju ubacivanje cijelog skupa podataka putem *for* petlje, dohvaćanje 5 zapisanih vrijednosti te uklanjanje cijelog skupa podataka, također s *for* petljom. Ove operacije, tokom testiranja, se kontinuirano izvode svakih 5 milisekundi što je vidljivo na programskom isječku 4.6.

Obzirom da se testiranje provodi za mali i veliki skup podataka, obje strukture podatka su testirane 30 puta za svaki skup. U konačnici, za testiranje podatkovnih struktura je izvedeno 120 testova. Vrijeme testiranja je postavljeno na 5 minuta s 1 sekundom intervala za uzimanje podataka.

Ispis 4.6 Programski dio za testiranje podatkovnih struktura

```
val testArrayMap = ArrayMap<String, String>()
val testHashMap = HashMap<String, String>()
var testDataStructure = "array"
var size = 200
findViewById<EditText>(R.id.test_name).setText(
"test_data_structure_"+testDataStructure+"map_"+size)
findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
RadioGroup.OnCheckedChangeListener { group, checkedId ->
    testDataStructure = ( when (checkedId) {
        R.id.arraymap -> "array"
        else -> "hash"
    })
    findViewById<EditText>(R.id.test_name).setText(
    "test_data_structure_"+testDataStructure+"map_"+size)
})
findViewById<RadioGroup>(R.id.radioGroup_size).setOnCheckedChangeListener(
RadioGroup.OnCheckedChangeListener { group, checkedId ->
    size = ( when (checkedId) {
        R.id.small -> 200
        else -> 2000
    })
    findViewById<EditText>(R.id.test_name).setText(
    "test_data_structure_"+testDataStructure+"map_"+size)
})

fixedRateTimer("timer", false, 0, 5) {
   if (recording) {
        if (testDataStructure.equals("array")){
            for (i in 1..size) { testArrayMap["Key$i"] = "Value$i" }
            testArrayMap["Key100"]
            testArrayMap["Key101"]
            testArrayMap["Key102"]
            testArrayMap["Key103"]
            testArrayMap["Key104"]
            for (i in 1..size) { testArrayMap.remove("Key$i") }
        } else if (testDataStructure.equals("hash")) {
            for (i in 1..size) { testHashMap["Key$i"] = "Value$i"  }
            testHashMap["Key100"]
            testHashMap["Key101"]
            testHashMap["Key102"]
            testHashMap["Key103"]
            testHashMap["Key104"]
            for (i in 1..size) { testHashMap.remove("Key$i") }
        }
    }
}
```

Rezultati će se uspoređivati na temelju veličine skupova podataka. Stoga, za manji skup od 200 elemenata, rezultati su prikazani tablicom 4.7 gdje su prikazane prosječne vrijednosti struje, napona, snage i ukupne energije. U ovom slučaju, može se primjetiti da korištenjem ArrayMap ili HashMap strukture nema osjetne razlike u energetskom utjecaju. Naime, energetska razlika između uporabe tih podatkovnih struktura iznosi svega 2 J. Također, za oba slučaja, potrošnja snage po satu se može smatrati identičnom i to u iznosu od 1 W. Dakle, s energetskog aspekta, nije bitno koja se struktura koristi za manji skup elemenata, no HashMap struktura ipak donosi minimalno veću energetsku uštedu. S druge strane, razmatrajući rezultate kod velikog skupa podataka od 2000 elemenata prikazani tablicom 4.8, razlika u potrošnji energije je značajna. Budući da je korištenje ArrayMap strukture potrošilo 201,85 J više nego HashMap struktura, zasigurno se može pretpostaviti da kod velikog skupa elemenata, HashMap podatkovna struktura treba biti izabrana.

Tablica 4.7 Rezultati testiranja podatkovnih struktura izraženi prosjekom od 30 testiranja po 5 minuta za mali skup podataka

Struktura	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
ArrayMap	254,57377	3,95810	1,00761	302,28328	6,05319
HashMap	252,03667	3,97065	1,00082	300,24736	7,89687

Tablica 4.8 Rezultati testiranja podatkovnih struktura izraženi prosjekom od 30 testiranja po 5 minuta za veliki skup podataka

Struktura	**Struja**	**Napon**	**Snaga**	**Ukupna energija**	
	$\overline{I}\,[mA]$	$\overline{U}\,[V]$	$\overline{P}\,[W]$	$\overline{E}\,[J]$	σ
ArrayMap	476,44791	3,95834	1,88576	565,72687	7,42531
HashMap	306,01177	3,96380	1,21291	363,87225	6,06813

Za oba skupa elemenata proveden je t-test. Kod manjeg skupa elemenata, analizom je potvrđeno da razlika u potrošnji energije između korištenja ArrayMap podatkovne strukture ($302{,}28 \pm 6{,}05$ J) i HashMap podatkovne strukture ($300{,}25 \pm 7{,}90$ J)

nije statistički značajna: t(29) = -1,025; p = 0,31 > 0,05. Kod većeg skupa podataka, utvrđeno je da je potrošnja energije koristeći ArrayMap podatkovnu strukturu ($565{,}73 \pm 7{,}43$ J) statistički značajno veća od potrošnje energije koristeći HashMap podatkovnu strukturu ($363{,}87 \pm 6{,}07$ J): t(29) = -123,116; p < 0,001.

Za kraj, može se zaključiti da ArrayMap strutkura doista troši više energije kod većeg skupa podataka, dok kod manjeg skupa podataka nema značajne razlike u potrošnji energije te se ne može utvrditi koja struktura ima veću potrošnju.

4.2.2 Korištenje dretvi

Korištenje dretvi u programiranju omogućuje paralelno izvršavanje više dijelova koda čime omogućuju aplikacijama da odgovore na više korisničkih zahtjeva istovremeno ili da obavljaju pozadinske zadatke bez utjecaja na korisničko sučelje [17]. Iako dretve optimiziraju izvedbu aplikacije, istovremeno mogu zahtijevati dodatnu potrošnju energije. Paralelno izvršavanje više dijelova koda zahtijeva resurse poput procesorskog vremena i memorije, što može rezultirati povećanom potrošnjom energije.

Za potrebe testiranja, simulirati će se rasčlanjivanje dugačkog odgovora u JSON formatu. U prvom slučaju, zadatak će se izvesti na glavnoj dretvi, dok u drugom slučaju napraviti će se nova dretva za izvršenje zadatka. Na takav način će se dobiti dojam o energetskog učinkovitosti dretvi, te u kojem slučaju i omjeru je povećana potrošnja energije.

Programskim isječkom 4.7 je prikazan programski kod kojim se, ovisno o odabiru dretvi, izvršava zadatak rasčlanjivanja odgovora u JSON formatu. Zadatak će se ili izvršiti na zadanoj glavnoj dretvi ili će se stvoriti nova dretva i pokrenuti izvršavanje. Izvršavanje zadatka se konstantno ponavlja svake milisekunde.

Za svaki test je postavljeno vrijeme izvođenja na 5 minuta s 1 sekundom intervala uzimanja podataka baterije. Obzirom da su testirana dva slučaja, a nad svakim slučajem je provedeno 30 testova, ukupan broj izvršenih testova iznosi 60.

Nakon testiranja, rezultati prikazani tablicom 4.9 pokazuju prosječne vrijednosti struje, napona, snage i ukupne energije za oba slučaja. Razmatrajući energiju, za-

sigurno se može primjetiti da stvaranje nove dretve za izvršavanje operacija iziskuje dodatno energije. Naime, u 5 minuta izvođenja, operacija izvršena na novostvorenoj dretvi je potrošila 234,32 J više energije. Istovremeno, potrošnja snage po satu je povećana i to za 0,78 W.

Ispis 4.7 Programski dio za testiranje dretvi

```
testName = "test_threads_withoutThread"
findViewById<EditText>(R.id.test_name).setText(testName)
findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
RadioGroup.OnCheckedChangeListener { _, checkedId ->
    when (checkedId) {
        R.id.withoutThread -> testName = "test_threads_withoutThread"
        R.id.withThread -> testName = "test_threads_withThread"
    }
    findViewById<EditText>(R.id.test_name).setText(testName)
})
fixedRateTimer("timer", false, 0, 1) {
    if (recording) {
        val checkedRadioButtonId =
            findViewById<RadioGroup>(R.id.radioGroup).checkedRadioButtonId
        when (checkedRadioButtonId) {
            R.id.withoutThread -> parseLargeJSONResponse()
            R.id.withThread -> {
                Thread {
                    parseLargeJSONResponse()
                }.start()
            }}
    }
}
```

Iako je očekivano da ostanak na glavnoj dretvi ušteđuje energiju, još uvijek je moguće dodatno istražiti razliku u potrošnji energije putem analize vremenskog trajanja baterije. Stoga, s ostankom na glavnoj dretvi, trajanje baterije iznosi 6 sati i 59 minuta. Uvođenjem nove dretve, trajanje baterije se smanjuje na 5 sati i 12 minuta. Time se dobiva razlika od skoro 2 sata, točnije 1 sat i 46 minuta.

Pomoću t-testa, ustanovljeno je da potrošnja energije stvarajući novu dretvu ($918{,}68 \pm 20{,}26$ J), statistički značajno je veća od potrošnje energije ostanka na glavnoj dretvi ($684{,}36 \pm 76{,}25$ J): $t(29) = 15{,}773$; $p = < 0{,}001$.

Zaključno, stvaranje nove dretve zahtijeva više energije no, ukoliko izvedba aplikacije pati zbog izvršavanja jednog ili više zadataka na istoj, glavnoj dretvi, svakako se preporuča uvođenje nove dretve. Prije nego što se uvede nova dretva, obzirom da

se radi o značajnoj razlici energije, preporučuje se pažljivo razmotriti dugotrajnost i složenost zadatka prije uvođenja nove dretve.

Tablica 4.9 Rezultati testiranja dretva izraženi prosjekom od 30 testiranja po 5 minuta

Dretva	**Struja** $\overline{I}\,[mA]$	**Napon** $\overline{U}\,[V]$	**Snaga** $\overline{P}\,[W]$	**Ukupna energija** $\overline{E}\,[J]$	σ
Nova dretva	776,76200	3,93821	3,06277	918,68057	20,25790
Glavna dretva	578,77657	3,94132	2,28120	684,36047	76,25394

4.2.3 Petlje

U programiranju, petlje predstavljaju ključni koncept za optimizaciju izvršenja koda. One omogućuju automatizaciju ponavljajućih radnji, što znači da se isti blok koda može izvršiti više puta s različitim vrijednostima ili uvjetima. Također, korištenje odgovarajuće vrste petlje može smanjiti potrošnju resursa, kao što su CPU ciklusi i memorija [18]. Na primjer, *for* petlje se koriste kada unaprijed znamo broj iteracija, dok se rekurzija koristi za rješavanje problema kroz samopozivajuće funkcije. Također, mnogi programski jezici imaju ugrađene funkcije ili metode koje su optimizirane za određene operacije, što može dodatno poboljšati performanse aplikacije. Ovdje se postavlja pitanje, kada odabir implementacije petlje ne utječe na izvedbu aplikacije, koju petlju odabrati kako bi se postigla energetska učinkovitost aplikacije.

U testiranju su uzete u obzir *for* petlja, rekurzija te ugrađene funkcije za rješavanje istog problema. U sva tri slučaja problem je isti - izračunati faktorijelu od broja 1000.

Na programskom isječku 4.8 je prikazan programski kod kojim se mijenja petlja za izvršavanje pomoću radio gumba. Izvršavanje zadatka računanja faktorijele će se izvršiti unutar 1 milisekunde, stoga, tijekom testiranja, neprekidno svake milisekunde, poziva se odgovarajuća funkcija koja sadrži pripadajuću petlju.

Ispis 4.8 Programski dio za testiranje petlji

```
testName = "test_loops_for"
findViewById<EditText>(R.id.test_name).setText(testName)
findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
RadioGroup.OnCheckedChangeListener { group, checkedId ->
    when (checkedId) {
        R.id.forLoop -> testName = "test_loops_for"
        R.id.recursion -> testName = "test_loops_recursion"
        R.id.builtIn -> testName = "test_loops_builtIn"
    }
    findViewById<EditText>(R.id.test_name).setText(testName)
})


fixedRateTimer("timer", false, 0, 1) {
    if (recording) {
        val checkedRadioButtonId =
        findViewById<RadioGroup>(R.id.radioGroup).checkedRadioButtonId

        when (checkedRadioButtonId) {
            R.id.forLoop -> factorialForLoop(1000)
            R.id.recursion -> factorialRecursion(1000)
            R.id.builtIn -> factorialBuiltIn(1000)
        }
    }
}
```

Funkcije za izračun faktorijele pomoću petlji su prikazane isječkom 4.9, za sva tri slučaja.

Svaki test je izvodio određenu petlju trajanju od 5 minuta uzimajući vrijednosti baterije svaku sekundu. Za sva tri slučaja se provelo 20 testova čime se dobiva 60 sveukupno izvršenih testova.

Prosječne vrijednosti struje, napona, snage i ukupne energije dobivene tijekom testiranja triju implementacija petlji prikazane su tablicom 4.10. Može se primjetiti da je računanje faktorijele rekurzijom potrošilo najviše energije od 560,15 J, dok se za izvođenje *for* petlje utrošilo najmanje energije od 550,24 J.

Međutim, razmatrajući potrošenu snagu za pojedinu petlju, može se uočiti da su približno jednake vrijednosti. U svakom slučaju, za kapacitet baterije od 15,96 Wh, računanje faktorijele s *for* petljom iscrpiti će bateriju za 16 sati i 34 minute. Implementacija s ugrađenim funkcijama će skratiti vrijeme trajanja baterije na 16

Ispis 4.9 Funkcije za izračun faktorijele putem for petlje, rekurzije i ugrađenih funkcija

```
1
2   fun factorialForLoop(n: Int): Long {
3       var result: Long = 1
4       for (i in 1..n) {
5           result *= i
6       }
7       return result
8   }
9   fun factorialRecursion(n: Int): Long {
10      if (n == 0 || n == 1) {
11          return 1
12      }
13      return n * factorialRecursion(n - 1)
14  }
15  fun factorialBuiltIn(n: Int): Long {
16      return (1..n).fold(1L) { acc, i -> acc * i }
17  }
```

Tablica 4.10 Rezultati testiranja petlji izraženi prosjekom od 20 testiranja po 5 minuta

Petlja	**Struja** $\overline{I}\,[mA]$	**Napon** $\overline{U}\,[V]$	**Snaga** $\overline{P}\,[W]$	**Ukupna energija** $\overline{E}\,[J]$	σ
***For* petlja**	246,00676	3,92276	0,96501	550,24035	89,93103
Ugrađene funkcije	245,96624	3,96169	0,97440	555,29477	89,74892
Rekurzija	247,06709	3,97790	0,98279	-560,15078	91,13047

sati i 22 minute. Sa rekurzijom će se baterija najbrže isprazniti i to za 16 sati i 14 minuta.

Istodobno, provedena je analiza ANOVA RM testom. Pritom je utvrđeno da postoji značajna razlika u potrošnji energije ovisno o implementaciji petlje: $F(2,38) = 8{,}386$; $p < 0{,}001$. Također, post-hoc analizom s međusobnim usporedbama, koristeći Bonferronijevu korekciju, utvrđeno je sljedeće:

- Potrošnja energije implementacijom pomoću *for* petlje ($550{,}24 \pm 89{,}93$ J) je statistički značajno manja od potrošnje energije implementacijom pomoću ugra-

đenih funkcija ($555{,}29 \pm 89{,}75$ J): $p = 0{,}002 < 0{,}05$.

- Potrošnja energije implementacijom pomoću *for* petlje ($550{,}24 \pm 89{,}93$ J) je statistički značajno manja od potrošnje energije implementacijom s rekurzijom ($560{,}15 \pm 91{,}13$ J): $p < 0{,}001$.
- Razlika u potrošnja energije kod implementacija pomoću ugrađenih funkcija ($555{,}29 \pm 89{,}75$ J) i rekurzije ($560{,}15 \pm 91{,}13$ J) nije statistički značajna: $p = 0{,}134 > 0{,}05$.

Računanje faktorijele broja 1000 nije zahtijevna operacija, te inače izbor korištenja petlje ovisi o složenosti zadatka te vremenu izvršavanja. No, u ovom slučaju, može se potvrditi da izvođenje iteracija pomoću *for* petlje koristi manje energije od ostalih implementacija testirane ovim radom.

4.2.4 Osluškivanje senzora

U razvoju Android aplikacije postoje resursi poput datoteka, medija, mrežnih veza i senzora koje se, nakon njihovog korištenja, mora otpusiti. Ukoliko nisu više potrebni, njihovo neotpuštanje može dovesti do curenja memorije i povećanja potrošnje energije [19].

U ovom slučaju, senzori će poslužiti za testiranje koliko energije njihovo osluškivanje troši. Provest će se testiranje s 4 senzora koji uključuju: akcelerometar, senzor blizine, kompas i senzor svjetla, Nadalje, svi senzori će dohvaćati informacije s najvišom frekvencijom od 0 mikrosekundi, odnosno što je brže moguće [19]. U jednom slučaju će svi senzori imati registriran osluškivač, dok će u drugom osluškivač biti odregistriran.

Prethodno opisano se može vidjeti na isječku 4.10 gdje se inicijaliziraju senzori te, ovisno o vrsti testa, registriraju ili odregistriraju osluškivači.

Ukupno je provedeno 30 testova, gdje je za svaki slučaj provedeno 15 testova. Vrijeme testiranja je postavljano na 30 minuta gdje se informacije o potrošnji računaju svake sekunde.

Tablicom 4.11 su prikazani rezultati s uključenim odnosno isključenim senzorima

putem prosječnih vrijednosti za struju, napon, snagu te ukupnu energiju. Sa potrošenom energijom od 1757,74 J unutar 30 minuta, može se utvrditi da ne osluškivanje senzora kada nisu potrebni uštađuje energiju. Budući da potrošnja energije iznosi 1804,58 J kada su sva 4 osluškivača registrirana, razlika u energiji se svodi na 46,8 J kroz 30 minuta izvođenja aplikacije. Iako se ta razlika može činiti neznatnom u trajanju od 30 minuta, dugoročno može izazvati nepotrebno iscrpljivanje baterije.

Ispis 4.10 Programski dio za testiranje slušanje senzora

```
val sensorManager : SensorManager = getSystemService(Context.SENSOR_SERVICE)
val accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
val proximity = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
val compass = sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION)
val light = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)

findViewById<RadioGroup>(R.id.radioGroup).setOnCheckedChangeListener(
RadioGroup.OnCheckedChangeListener { group, checkedId ->
    when (checkedId) {
        R.id.sensorOn -> {
            testName = "test_sensors_On"
            sensorManager.registerListener(
            sensorListener, accelerometer, SensorManager.SENSOR_DELAY_FASTEST)
            sensorManager.registerListener(
            sensorListener, proximity, SensorManager.SENSOR_DELAY_FASTEST)
            sensorManager.registerListener(
            sensorListener, compass, SensorManager.SENSOR_DELAY_FASTEST)
            sensorManager.registerListener(
            sensorListener, light, SensorManager.SENSOR_DELAY_FASTEST)
        }
        R.id.sensorOff -> {
            testName = "test_sensors_Off"
            sensorManager.unregisterListener(null)
        }
    }
    findViewById<EditText>(R.id.test_name).setText(testName)
})
```

Vremenski aspekt trajanja baterije ukazuje na razliku od 25 minuta uštede. Naime, sa snagama od 0,98 W i 1 W, registriranje osluškivača senzora ispraznit će bateriju kroz 15 sati i 55 minuta, dok uklanjanje osluškivača povećava trajanje na 16 sati i 20 minuta.

Potrošene energije su analizirane pomoću t-testa. Rezultatima je utvrđeno da je potrošnja energije s neregistriranim osluškivačem senzora ($1757{,}74 \pm 45{,}25$ J) sta-

Tablica 4.11 Rezultati testiranja senzora izraženi prosjekom od 15 testiranja po 30 minuta

Osluškivač	**Struja** $\overline{I}\,[mA]$	**Napon** $\overline{U}\,[V]$	**Snaga** $\overline{P}\,[W]$	**Ukupna energija** $\overline{E}\,[J]$	σ
Neregistriran	248,63815	3,92796	0,97653	1757,74322	45,25492
Registriran	256,56153	3,90770	1,00255	1804,58309	50,71525

tistički značajno manja od potrošnje energije s registriranim osluškivačem senzora ($1804{,}58 \pm 50{,}72$ J): $t(14) = 2{,}841$; $p = 0{,}013 < 0{,}05$.

Za kraj, može se potvrditi da ne oslobađanje senzora nakon upotrebe utječe na energiju. Ako se osluškivanje kontinuirano izvodi, bilo u aktivnoj ili pozadinskoj aplikaciji, ta dodatna potrošnja energije može biti kumulativna i dovesti do značajnog smanjenja trajanja baterije. Stoga je važno pažljivo upravljati takvim osluškivanjem kako bi se osigurala optimalna iskoristivost resursa i energetska učinkovitost.

Poglavlje 5

Zaključak

Kroz ovaj rad testirali su se osnovni razvojni principi Android aplikacija, od implementiranja korisničkog sustava do izvornog programskog koda aplikacije. Kao rezultat, dobivene su potvrđene temeljne smjernice za razvoj energetski učinkovitih Android aplikacija. Prvenstveno se može potvrditi da kod dizajniranja korisničkog sučelja, korištenje tamnijih boja rezultira manjoj emitiranoj svjetlosti te ujedno smanjenoj potrošnji energije. Također, treba paziti na hijerahiju elemenata te ju držati što plosnatijom. Drugim riječima, treba računati na dubinu ugniježđenih rasporeda. Premda sjene na elementima dodaju trodimenzionalan izgled, treba ih koristit umjereno zbog mogućeg povećanja potrošnje energije. Između ostalog, prilikom odabira formata slike za uporabu u aplikaciji, treba imati na umu da SVG format koristi više energije nego JPG ili PNG format.

Što se tiče programskog dijela aplikacije, energija se može uštediti na razne načine. Ukoliko postoji veliki skup podataka, on se može realizirati pomoću HashMap strukture podataka. Naime, između HashMap i ArrayMap strukture, dokazano je da ArrayMap nije pogodan za korištenje s velikim skupovima. Glede malog skupa, obje strukture imaju približne energetske rezultate, stoga je potrebno osloniti se na druge karakteriste i perfomanske ovih struktura. Osim načina spremanja podataka, potrebno je znati kako energetski učinkovito izvršavati određene zadatke. Shodno tome, stvaranje nove dretve za izvršavanje zadatka iziskuje znatno više energije nego ostavljanje zadatka na glavnog dretvi. Pri tome treba pripaziti riskira li se izvedba aplikacije i narušava li se korisničko iskustvo ostavljanjem zadatka na glavnog dre-

tvi. Također, određene zadatke koji se riješavanju iteracijama, bolje ih je realizirati pomoću *for* petlje. Uz sve navedeno, treba računati na oslobađanje resursa ukoliko nisu potrebni. Pritom se prvenstveno podrazumijeva oslobađanje senzora odnosno odregistracija osluškivača senzora. Dokazano je da kontinuirano slušanje senzora kada nije potreban utječe negativno na energiju.

Primjenom prethodnih savjeta za razvoj korisničkog sučelja, moguće je postići pasivnu energetsku uštedu. Također, pridržavanje smjernica za programski dio aplikacije omogućuje uštedu energije, s time da treba računati na izvedbu aplikacije. Razvijanje aplikacije se uvijek može voditi po ovim smjernicama te, ukoliko je to moguće, navoditi programere da iskoriste energetski učinkovite implementacije.

Bibliografija

[1] F. L. Statista, "Market share of mobile operating systems worldwide 2009-2022," https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/, 2022, s interneta; pristupljeno 20.10.2022.

[2] Google. (2022) Meet android studio. , s Interneta, https://developer.android.com/studio/intro S interneta; pristupljeno 30.10.2022.

[3] T. R. Emteria. (2022) A brief history of android studio. , s Interneta, https://emteria.com/learn/history-android-studio/ S interneta; pristupljeno 30.10.2022.

[4] w3schools.com. (2022) Kotlin introduction. w3schools.com. , s Interneta, https://www.w3schools.com/kotlin/kotlin_intro.php S interneta; pristupljeno 30.10.2022.

[5] Google. (2022) Batterymanager. Google. , s Interneta, https://developer.android.com/reference/android/os/BatteryManager S interneta; pristupljeno 30.10.2022.

[6] GSMArena. (2022) Huawei p40 lite. GSMArena. , s Interneta, https://www.gsmarena.com/huawei_p40_lite-9996.php S interneta; pristupljeno 30.10.2022.

[7] Rapidtables. (2022) mah u wh kalkulator pretvorbe. , s Interneta, https://www.rapidtables.org/hr/calc/electric/mah-to-wh-calculator.html S interneta; pristupljeno 31.10.2022.

[8] F. Bouaffar, O. Le Goaer, and A. Noureddine, "Powdroid: Energy profiling of android applications," pp. 251–254, 11 2021, s interneta; pristupljeno 15.10.2022.

[9] A. L. System. (2013) Power capacity and power capability. , s Interneta, https://learn.adafruit.com/all-about-batteries/power-capacity-and-power-capability S interneta; pristupljeno 14.6.2023.

[10] S. P. K. B. Programming. (2019) Android ui performance. Better Programming. , s Interneta, https://betterprogramming.pub/android-ui-performance-11b57ac4af8c S interneta; pristupljeno 6.1.2023.

[11] L. T. ZDNet. (2018) Google: Here's why dark mode massively extends your oled phone's battery life. ZDNet. , s Interneta, https://www.zdnet.com/article/google-heres-why-dark-mode-massively-extends-your-oled-phones-battery-life/ S interneta; pristupljeno 15.1.2023.

[12] Google. (2023) Optimizing view hierarchies. Google. , s Interneta, https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies S interneta; pristupljeno 2.2.2023.

[13] ——. (2023) Overdraw. Google. , s Interneta, https://developer.android.com/topic/performance/rendering/overdraw S interneta; pristupljeno 18.2.2023.

[14] M. Documentation. (2023) Image file type and format guide. , s Interneta, https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types S interneta; pristupljeno 18.2.2023.

[15] K. E. Joyce. (2020) Why understanding data structures is so important to coders. The Pragmatic Programmers LLC. , s Interneta, https://www.techtarget.com/searchdatamanagement/feature/Why-understanding-data-structures-is-so-important-to-coders S interneta; pristupljeno 15.6.2023.

[16] ANDROIDANDIOT. (2015) Android performance pattern - arraymap vs hashmap. , s Interneta, https://androidandbrillo.wordpress.com/2015/09/07/android-performance-patterns-arraymap-vs-hashmap/ S interneta; pristupljeno 15.6.2023.

[17] Google. (2023) Better performance through threading. Google. , s Interneta, https://developer.android.com/topic/performance/threads S interneta; pristupljeno 20.6.2023.

[18] N. Bhargav. (2022) Recursion and looping. Baeldung. , s Interneta, https://www.baeldung.com/cs/recursion-looping S interneta; pristupljeno 27.6.2023.

[19] Google. (2023) Sensors overview. , s Interneta, https://developer.android.com/guide/topics/sensors/sensors_overview S interneta; pristupljeno 1.6.2023.

Sažetak

U ovome radu su analizirani različiti aspekti razvoja mobilnih aplikacija za Android OS, pri čemu se u fokus stavlja njihov utjecaj na potrošnju baterije mobilnog uređaja odnosno energetska učinkovitost. Implementirana je ispitna aplikacija za mjerenje utroška energije koja je iskorištena u eksperimentalnim mjerenjima za prethodno definirane scenarije. Testnim scenarijima se, s aspekta energetske učinkovitosti, analiziraju različiti aspekti u oblikovanju korisničkog sučelja i pisanju programskog koda. Na temelju dobivenih rezultata moguće je identificirati problematične pristupe implementaciji mobilnih aplikacija koji negativno utječu na potrošnju energije i očekivano trajanje baterije.

Ključne riječi **— energetska učinkovitost, Android, mobilne aplikacije, trajanje baterije, korisničko sučelje**

Abstract

This thesis analyzes different aspects of mobile application development for Android OS, focusing on their impact on mobile device battery consumption, i.e. energy efficiency. A test application for measuring energy consumption has been implemented and utilized in experimental measurements for previously defined scenarios. In the test scenarios, various aspects of the user interface design and the creation of the program code are analyzed from the point of view of energy efficiency. The results obtained can be used to identify problematic approaches in the implementation of mobile applications that have a negative impact on energy consumption and the expected battery life.

Keywords **— energy efficiency, Android, mobile applications, battery life, user interface**