Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004:007
DOI: https://doi.org/10.24867/23OI01Zvekic

POREĐENJE PERFOMANSI IZMEĐU ANDROID APLIKACIJA RAZVIJENIH UPOTREBOM JETPACK COMPOSE ALATA I UPOTREBOM XML JEZIKA

PERFORMANCE COMPARISON BETWEEN ANDROID APLIKACIONS DEVELOPED USING JETPACK COMPOSE TOOL AND XML LANGUAGE

Nataša Zvekić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INFORMACIONO-KOMUNIKACIONI SISTEMI

Kratak sadržaj – *Ovaj rad predstavlja upoređivanje performansi Android aplikacija koje za razvoj korisničkog interfejsa koriste različite alate, i to Jetpack-Compose alat i XML programski jezik.*

Ključne reči: *Android, Kotlin, korisnički interfejs, perfomanse, APK*

Abstract – *This paper presents performance comparison between Android applications that uses different tools for building user interface, and that is JetpackCompose tool and XML programing language.*

Keywords: *Android, Kotlin, user interface, performance, APK*

1. UVOD

Upotreba mobilnih tehnologija postala je trend kao i obavezni alat u svakodnevnom životu. Mobilni uređaji sadrže brojne aplikacije koje se odnose na različite kategorije kao što su zabava, zdravlje, životni stil, itd, čime su korisne za brojne različite potrebe. Uspeh neke aplikacije zavisi od toga koliko je korisnik upotrebljava, odnosno od njene upotrebljivosti i koliko odgovara korisničkim zahtevima u zavisnosti od prethodnog iskustva [**1**]. Važnost korisničkog interfeja postala je veća sa povećanjem broja aplikacija i njihovih korisnika. Istovremeno, korisnici imaju manje volje da komuniciraju sa teškim ili neprijatnim korisničkim interfejsom [**2**].

Prema [**3**] performanse se ističu kao odlična karakteristika jer doprinosi svim korisnicima u svim scenarijima, te je potrebno dizajnirati zadatke tako da mogu da odreaguju, daju odličnu povratnu informaciju i izbegavaju nepotrebna čekanja.

Kada se govori o performansama mobilnih aplikacija, govori se o brzini učitavanja korisničkog interfejsa, o brzini reagovanja aplikacije na korisničke zahteve (klikove, unos teksta,..), potrošnju baterije, brzinu ažuriranja aplikacije, itd.

Cilj ovog rada jeste da na praktičnom primeru izrađenih Android mobilnih aplikacija uporedi performanse korisničkog interfejsa kada je u pitanju vreme potrebno da korisnik otpočne interakciju sa aplikacijom.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Marko Arsenović, docent.

Upoređivaće se performanse aplikacije čiji je korisnički interfejs izgrađen upotrebom proširivog opisnog (XML, eng. *Extensible Markup Language)* jezika i aplikacije gde je korisnički interfejs izgrađen upotrebom *Jetpack Compose* alata.

S obzirom da je *Jetpack Compose* alat izgrađen sa idejom da obezbedi bolje performanse, manje koda potrebnog za razvoj korisničkog interfejsa i ubrza proces razvoja aplikacija, ovaj rad je motivisan utvrđivanjem da li postoje razlike između ova dva načina kreranja korisničkog interfejsa i, ako postoje, koliko su one značajne.

Za potrebe istraživanja u ovom master radu razvijene su dve Android aplikacije, gde jedna koristi *Jetpack Compose* alat, a druga XML jezik, za razvoj korisničkog interfejsa i upoređivaće se vreme koje je potrebno korisničkom interfejsu dok ne uđe u fazu interakcije sa korisnikom, a koje je mereno i predstavljeno u milisekundama.

Pored toga, upoređivaće se i veličine Android paket fajlova (APK, eng. *Android Package Kit*) ove dve aplikacije predstavljene u megabajtima, s obzirom da u današnje vreme korisnici na svojim mobilnim uređajima čuvaju veći broj aplikacija, dok je memorija telefona za skladištenje istih ograničena, a u nekim slučajevima i nedovoljna za sve željene aplikacije.

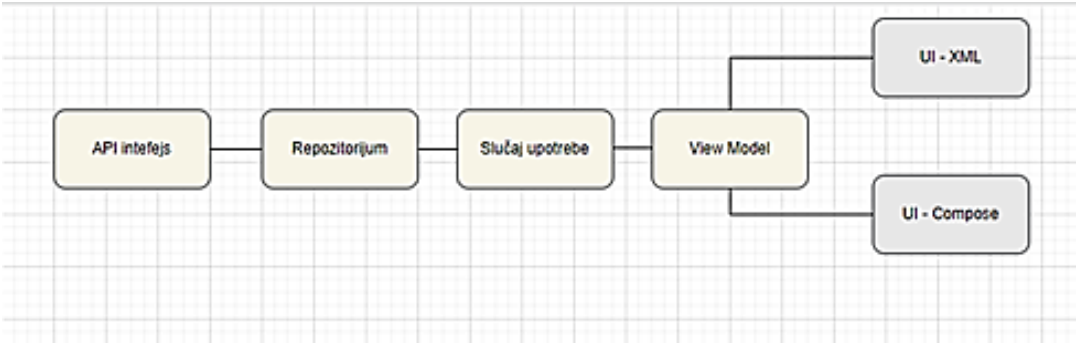
Ovim istraživanjem se ujedno daje odgovor i na pitanja da li je isplativo migrirati postojeće aplikacije napisane XML jezikom i da li je isplativo učiti novi alat za razvoj korisničkog interfejsa.

Interfejsi mobilnih aplikacija su vremenom postali interesantniji za upotrebu jer obezbeđuju funkcionalnosti koje nisu dostupne na desktop uređajima, kao što su na primer globalni sistem za pozicioniranje (GPS, eng. *Global Positioning System)* ili akcelerometar, koji omogućavaju da se ponašanje aplikacije prilagodi poziciji u prostoru ili drugim parametrima. Na ovaj način, korisnički interfejs mobilnih uređaja je u mogućnosti da automatski odabere šta i kako da prikaže na ekranu, zavisno od toga šta se dešava sa korisnikom i sa njegovom okolinom [**4**].

Tradicionalni način za izgradnju korisničkog interfejsa Android aplikacija podrazumeva deklarisanje elemenata upotrebom XML jezika. Ovaj način kreiranja korisničkog interfejsa, baziranog za definisanju pogleda (eng. *view*) kao gradivnih blokova, je od strane programera lako prihvaćen jer je dosta intuitivan i jer je XML kao jezik dosta godina unazad poznat i primenjivan.

2. OPIS MERENJA

Merenje u ovom radu je sprovedeno tako što su napravljene dve aplikacije sa identičnom poslovnom logikom i identičnim korisničkim interfejsom, gde je korisnički interfejs prve aplikacije izgrađen upotrebom XML programskog jezika, dok kod druge aplikacije upotrebom Jetpack Compose alata.



Slika 1. *Arhitektura aplikacije*

2.1. Opis aplikacija

Aplikacije koje su se koristile za merenje služe za prikaz deviznih kurseva koji su grupisani po tipovima, gde se za svaki tip prikazuje lista pripadajućih kurseva, dok se za svaki kurs prikazuje naziv, jedinica i vrednost. Ovakav prikaz liste u listi je izabran iz razloga što je to dosta kompleksan prikaza korisničkog interfejsa upotrebom XML programskog jezika, jer podrazumeva definisanje nekoliko XML fajlova i dva adaptera koji popunjavaju liste, kako bi se dobio željeni prikaz, zbog čega zahteva i određeno vreme za obradu, dok se upotrebom Jetpack Compose alata ovakav prikaz može dobiti prostim pozivanjem pomerajućih lista u redu koje se pozivaju u okviru pomerajuće liste u koloni.

Iznad ovih lista se nalazi polje za unos gde korisnik ima mogućnost unosa određenog teksta po kom će se filtrirati date liste po nazivu valuta. Drugi deo aplikacije obuhvata dve padajuće liste kojima se bira prva valuta koja će se konvertovati u selektovanu valutu, i pruža uvid u kretanje kursa u poslednja 24 časa.

Podaci koji se koriste za prikaz preuzeti su sa javnih pristupnih tački (eng. endpoint) i to

https://api.coingecko.com/api/v3/exchange_rates

za dobavljanje liste deviznih kurseva i

https://cex.io/api/price_stat/{prvaValuta}/{drugaValuta}

gde se u telu poziva šalju vrednosti za koji vremenski period unazad su potrebni podaci i koji broj podataka se očekuje, što je u primeru ove aplikacije u poslednjih 24 časa i to poslednjih deset zapisa.

Za obe aplikacije se na potpuno isti način dobavljaju podaci, transformišu i puštaju do dela sa korisničkim interfejsom, gde se isti preuzimaju i obacuju u odgovarajuća polja. Ono što je takođe zajedničko za obe aplikacije jeste da sadrže po jednu glavnu aktivnost u koju je, radi pojednostavljenja merenja performansi, smešten kompletan sadržaj korisničkog interfejsa. Dalje, za prvu aplikaciju napisanu upotrebom XML jezika u okviru XML fajla aktivnosti se smeštaju View komponente koje će se prikazivati, dok se u drugoj aplikaciji, napisanoj upotrebom Jetpack Compose alata, za sadržaj aktivnosti poziva odgovarajuća composable funkcija.

Za dobavljanje podataka sa navedenih ruta korišćena je biblioteka Retrofit, koja čita i parsira podatke koji se transformišu u oblik, odnosno model, koji je definisan u aplikaciji. Ovaj interfejs koji izvršava pozive poziva repozitorijum, koji uzima pročitane podatke i vrši transformaciju nad njima, gde najpre proverava da li je poziv bio uspešan ili ne. Referencu na repozitorijum poseduje klasa koja predstavlja konkretan slučaj upotrebe, a to je dobavljanje i vraćanje podataka, i koja ima zadatak da samo izvrši poziv i vrati odgovor, gde je oslobođena od poslovne logike.

Referencu na klasu koja upravlja slučajevima upotrebe čuva klasa ViewModel koja je zadužena za čuvanje stanja koja će prikazivati na korisničkom interfejsu, od strane kog je i pozvana.

2.2. Način prikupljanja podataka

U obe aplikacije je mereno i upoređivano vreme potrebno da se korisnički interfejs prikaže korisniku i postane interaktivan, pri čemu se koristila generička metoda u androidu onResume, koja, kada se izvrši, označava da je korisnički interfejs spreman za interagovanje sa korisnikom. Ova metoda se nalazi u životnom ciklusu android aktivnosti i vreme je mereno tako što se štoperica završava u trenutku kada se uđe u datu metodu, dok je vreme mereno u milisekundama.

Kako bi se dobili što verodostojniji rezultati, obe aplikacije su bile pokrenute na sedam različitih android mobilnih uređaja, i to: Pixel 2, Pixel 2 XL, Pixel 3 XL, Pixel 4 XL, Pixel 5, Pixel 6 XL, Nexus 6P i Nexus One, sa devet različitih nivoa aplikativnog programskih interfejsa, od nivoa 23, koji je minimalni nivo koji podržava biblioteke za alat Compose, do nivoa 32, izuzev nivoa 28. Drugim rečima, aplikacije su pokretane na uređajima sa nivoom aplikativnog programskog interfejsa 21, 22, 23, 24, 25, 26, 27, 29, 30, 31 i 32, koji pokrivaju operativne sisteme, redom, 6, 7, 7.1, 8, 8.1, 10, 11, 12 i 13.

Pored toga, aplikacija je pokretana na simulatoru, umesto na prvom fizičkom uređaju, kako bi se postiglo da svi telefoni imaju pojednak broj instaliranih aplikacija, odnosno da su na svim telefonima pristutne samo sistemske android aplikacije, kako bi se postigli jednaki uslovi pri pokretanju aplikacija na različitim uređajima.

Na ovaj način bilo je moguće upotrebiti da li se vreme koje je mereno razlikuje u zavisnosti od nivoa operativnog sistema, odnosno od nivoa aplikativnog programsko interfejsa, kada se posmatraju obe aplikacije. Na svakom uređaju je svaka aplikacija pokretana tačno četiri puta, gde se od toga uzelo prosečno vreme koje se koristilo za dalje upoređivanje i donošenje zaključaka.

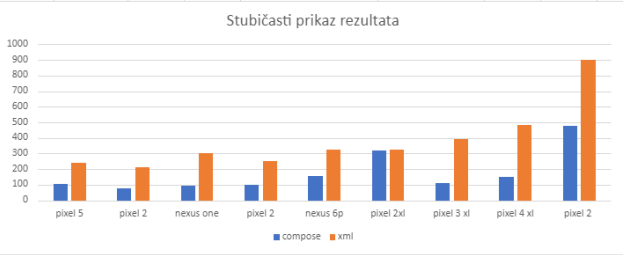
Drugim rečima, u ovom merenju dobijena je matrica od devet redova i pet kolona, gde se za sedam različitih uređaja, devet različitih nivoa aplikativnog programskog interfejsa i devet različitih operativnih sistema mereno vreme potrebno aplikaciji za interakciju sa korisnikom, za aplikacije čiji je korisnički interfejs kreiran upotrebom XML jezika i upotrebom Jetpack alata.

Kada se radi o drugom delu merenja, gde se upoređuju veličine APK fajlova, za obe aplikacije je izgenerisan APK fajl i upoređene su ove dve veličine predstavljene u megabajtima.

2.3. Rezultati istraživanja

Tabela 1. *Prosek rezultata merenja vremena*

Phone	OS	API level	Compose/ time	XML/time
Pixel 5	6	23	107	239
Pixel 2	7	24	76.25	210
Nexus One	7.1	25	94.25	304.25
Pixel 2	8	26	101.25	252.25
Nexus 6P	8.1	27	155.25	323.5
Pixel 2XL	10	29	320.25	323.75
Pixel 3XL	11	30	111	390
Pixel 4XL	12	31	152	482.25
Pixel 2XL	13	32	477.5	899.25



Slika 2. *Stubičasti prikaz rezultata merenja vremena*

Tabela 2. *Rezultati merenja veličine APK fajla*

UI	APK size in MB
Compose	8,8
XML	5,7

2.3.1. Analiza rezultata merenja

Na osnovu sprovedenog istraživanja, koje se prikazano tabelarnim i grafikonskim putem, jasno se može videti da postoji razlika kada se radi o vremenu potrebnom za spremnost korisnisničkog interfejsa za interakciju sa korisnikom između ove dve aplikacije. Pre svega, uviđa se razlika i kada se radi o načinu razvoja korisničkog interfejsa, i kada se radi o različitim operativnim sistemima. Pre svega, ono je odmah uočljivo jeste da je, kada se posmatra procentualna razlika, vreme mereno u aplikaciji gde se koristio JetpackCompose u većini slučajeva duplo kraće od vremena izmerenog u aplikaciji gde se koristio XML, dok je u nekim slučajevima kraće do čak 251 procenat.

Kada se uzmu u obzir posmatrani operativni sistemi i nivoi aplikativnog programskog interfejsa, jasno se vidi da najveću procentualnu razliku prave nivoi 25 i 30 i to, redom, 222% i 251%, dok se kod nivoa 29 primeti najmanja razlika u vremenu, koja je oko jednog procenta. Kada se radi o svim ostalim nivoima, primeti se tendencija da je ta razlika okvirno oko sto procenata.

S obzirom da su nivoi aplikativnog programskog interfejsa sortirani po vremenskoj hijerarhiji njihovog nastanka, tako posmatrani ukazuju da je na ranijim nivoima, koji su ovde uzeti u obzir, trebalo znatno manje vremena za početak interakcije korisničkog interfejsa sa korisnikom u odnosu na najnovije verzije, koje su prisutne u trenutku pisanja ovog rada i koje su uzete u obzir. Ovaj trend se primeti kod obe aplikacije. Kada se posmatra prva aplikacija, razvijena upotrebom JetpackCompose alata, razlika između najkraćeg vremena izmerenog na operativnom sistemu 7, koje je zabeležilo vreme 76,25 milisekundi i najdužeg vremena, zabeleženog na operativnom sistemu 13 – 477%, je čak 526% i ovo je najveća razlika uočena za prvu aplikaciju.

Kada se radi o drugoj aplikaciji gde je korišćen XML, minimalno zabeleženo vreme je takođe na operativnom sistemu 7, i to 210 milisekundi, a najveće zabeleženo vreme je 899,25 milisekundi na operativnom sistemu 13, dok je procentualna razlika između njih 328%. Ovako posmatrano, najkraće vreme za obe aplikacije je zabeleženo na operativnom sistemu 7, dok je najveće vreme zabeleženo na operativnom sistemu 13.

Drugi deo ovog istraživanja se odnosio na upoređivanje veličine APK fajlova za obe aplikacije koje se predstavljenje u megabajtima. U odnosu na merenje vremena za generisanje korisničkog interfejsa, kada se posmatra veličina APK fajla ove dve aplikacije, vidi se da aplikacija koristi Jetpack Compose zauzima 8,8 megabajta dok druga aplikacija, koja koristi XML, zauzima 5,7 megabajta, gde je prvi APK fajl veći za čak 54,38%.

3. ZAKLJUČAK

Na osnovu zvanične dokumentacije [**5**] ovaj master rad i sprovedeno istraživanje bilo je motivisano da dokaže da razvoj korisničkog interfejsa upotrebom JetpackCompose alata nudi brojne benefite koji su uočljivi kada su u pitanju brzina razvoja i memorija koju aplikacija zauzima, u odnosu na tradicionalni i godinama prisutan načina razvoja korisničkog interfejsa koji je podrazumeo korišćenje XML jezika. Na potpuno jednak način je mereno vreme za obe aplikacije, merenje je sprovedeno uz nekoliko ponavljanja, nad više različitih uređaja koji pokrivaju različite operativne sisteme.

Kao što je i pretpostavljeno pre pisanja ovog rada, generisanje korisničkog intefejsa upotrebom JetpackCompose alata zaista uzima manje vremena nego što se to dešava tradicionalnim načinom gde se koristi XML jezik za kreiranje korisničkog interfejsa.

Tradicionalni raspored pogleda je dosta izražajniji kada se izvršava merenje rasporeda, što olakšava da se kroz rasporede prođe više puta. Ovi višestrukti prolazi mogu prouzrokovati eksponencijalni rad ako se izvršavaju nad ugnježdenim tačkama u hijerarhiji prikaza.

Pored toga, sistem pogleda treba da veže XML raspored za odgovarajući ekran kada se prikazuje prvi put, dok kod Jetpack Compose alata ovaj trošak ne postoji jer su svi rasporedi pisani u programskom jeziku Kotlin i kompajliraju se na isti način kao i ostatak aplikacije [**6**].

Jetpack Compose primenjuje samo jedan prolazak kroz raspored za sve rasporede kreirane putem API ugovora (eng. *contracts*), što omogućava da se na efikasan način upravlja stablima korisničkog interfejsa sa velikim brojem članova. Ako je potrebno da se izvrši više merenja, Compose ima poseban sistem sa unutrašnjim merenjima [**6**].

Drugim rečima, Compose ne dozvoljava merenje u više prolaza, što znači da elementi rasporeda ne mogu izmeriti više od jednog puta svoje podređene kompnente kako bi isprobali razltčitu konfiguraciju merenja [**7**].

Pored toga, za aplikacije koje se koriste u ovom istraživanju odabran je jedan od verovatno najkompleksnijih komponenata korisničkog intefejsa, a to je lista u listi. Da bi se ovakav prikaz ostavio upotrebom XML jezika, potrebno je razviti dva RecyclerView prikaza koji emituju listu elemenata, s tim što je jednu listu potrebno ugnjezditi u drugu.

Ovo praktično znači da je potrebno kreirati dva XML fajla od kojih jedan predstavlja roditeljsku listu, a drugi prikaz liste koja se ugnježdava, zatim još jedan XML fajl koji opisuje elemente ugnježdene liste. Da bi se ovo ostvarilo, potrebno je definisati i dve klase koje se nazivaju adapterima i koje primaju liste i zadužene za prikaz samih lista, a pored toga i dve klase koje su zadužene da XML fajlove popune odgovarajućim vrednostima. Ovako rečeno, za ovakav prikaz komponente korisničkog intefejsa u XML jeziku potrebno je definisati tri XML fajla i četiri klase.

Nasuprot ovom načinu, kada se ovakva komponentu prikazuje upotrebom JetpackCompose alata, potrebno je definisati kako će izgledati komponenta ugnježdene liste, a zatim definisati kolonu koja će za svoje elemente iscrtavati redove koji primaju listu koju iscrtavaju. Dakle, u ovom slučaju je potrebno definisati samo tri Composable funkcije. Svaka komponenta koja se iscrtava, kao što je na primer dugme ili teksutalno polje, zahteva alociranje određene memorije, eksplicitno praćenje stanja i različite funkcije koje podržavaju sve slučajeve upotrebe. Pored toga, ručno kreirani pogledi zahtevaju eksplicitnu logiku koja će upravljati ponovnim iscrtavanjem ukoliko je to potrebno.

Jetpack Compose ovo rešava na nekoliko načina, pre svega ne zahteva objekte za ažuriranje pri iscrtavanju pogleda, elementi korisničkog interfejsa su funkcije čije informacije su zapisane u kompoziciji na način koji se može reprodukovati. Ovo pomaže da se smanji eksplicitno praćenje stanja, alociranje memorije i funkcija sa povratnim pozivima samo na one funkcije kojima su ta svojstva potrebna, nasuprot korišćenju svih funkcija datom view tipa kao što je to slučaj bez Compose alata [**6**].

Kada se radi o upoređivanju veličine APK fajla za ove dve aplikacije, jasno je da aplikacija koja koristi Jetpack Compose zauzima za 54,38% više prostora. Razlog za ovo jeste što prva aplikacija, koja koristi Compose koristi veći broj biblioteka kao zavisnosti koje se koriste kako bi se izgradio korisnički interfejs, gde je velik broj biblioteka vezan baš za upotrebu Compose alata. Što je veći broj klasa, metoda i zavisnosti koje se koriste – veća je veličina APK fajla.

4. LITERATURA

[1] Khan Kalimullah and Donthula Sushmitha, "Influence of Design Elements in Mobile Applications on User Experience of Elderly People".

[2] ANDREAS RIEGLER AND CLEMENS HOLZMANN, "Measuring Visual User Interface Complexity of Mobile Applications With Metrics".

[3] Everett N McKay, *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*.

[4] Luca Chittaro, "Designing Visual User Interfaces for Mobile Applications".

[5] [Online, poslednji put prostupano 19.10.2022.]. https://developer.android.com/jetpack/compose

[6] [Online, poslednji put pristupano 19.10.2022.]. https://developer.android.com/jetpack/compose/ergonomics

[7] [Online, poslednji put pristupano 23.10.2022.] https://developer.android.com/jetpack/compose/layouts/custom#create-custom

Kratka biografija:

Nataša Zvekić rođena je u Novom Sadu 1998. godine. Diplomski rad na Fakultetu tehničkih nauka iz oblasti Informaciono-komunikacionih sistema odbranila je 2021.god.

kontakt: zvekicnatasa123@gmail.com