Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/02BE37Skrbic

RAZVOJ APLIKACIJE ZA TRENUTNU RAZMENU PORUKA NA ANDROID SISTEMIMA U PROGRAMSKOM JEZIKU KOTLIN

FRAMEWORK INSTANT CHAT MESSAGES APPLICATION IMPLEMENTATION FOR ANDROID PLATFORM USING KOTLIN PROGRAMMING LANGUAGE

Nikola Škrbić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena implementacija aplikacije za trenutnu razmenu poruka u realnom vremenu. Takođe rad opisuje i korišćenje Kotlin programoskog jezika na Android platformi kao i praktičnu primenu same aplikacije. Sama implementacija i način funkcionisanja su takođe detaljnije opisani kroz poglavlja kako bi se dočarala kompletna slika sistema.*

Ključne reči: Android, čet, aplikacija, softver, Kotlin, Firebase.

Abstract – *–This paper represents a study about chat application for instant message communication in real time on Android platform. It also describes a Kotlin programming language and its usage in implementation on Android platform as well as practical usage of given application. Concrete implementation and functionallity of the application is also included in this work and is further elaborated in coming chapters in purpose of supplementing it for better complete image of whole system.*

Keywords: *Android, chat, application, software, kotlin, Firebase.*

1. UVOD

Mobilna aplikacija je program koji je namenjen prenosnim uređajima kao što su pametni telefoni, tableti i pametni satovi. Prvobitno su služile za obavljanje osnovnih funkcija, ali kako je tehnologija sve više napredovala, dobile su sasvim novi oblik. Danas, mobilne aplikacije su postale svakodnevnica i koriste se u različite svrhe. Kalkulator, video igra, kompas, online prodavnica su neke od mobilnih aplikacija. Izrada aplikacija postala je kompleksinija pa se dosta pažnje počelo posvećivati njihovim arhitekturama. Kako bi se arhitektura uklopila u samu izradu, potrebno je poštovati komponente operativnog sistema na kom se ona primenjuje, samim tim i životni vek aplikacije koja se izvršava na mobilnom uređaju. Treće poglavlje će se baviti Kotlin programskom jeziku [2]. prednosti korišćenja, takođe će biti više reči i o specifikaciji samog programskog jezika. Kroz četvrto poglavlje govoriće se o novim tehnologijama, Google Firebase, Ubrizgavanje zavisnosti i Gradle sistem automatizacije. Peto poglavlje govoriće o samoj implementaciji aplikacije. Biće detaljnije objašnjen model podataka koji se koristi u aplikaciji, način prijave na sistem, arihitektura sistema, komunikacija sa udaljenim servisima, kao i parcijalno učitavanje podataka i načinu testiranja istog. Poslednje poglavlje biće posvećeno zaključku ovog istraživanja i projekta.

2. ANDROID SISTEM

Android [1] je mobilni operativni sistem kompanije Google zasnovan na Linuks operativnom sistemu i njegovom jezgru, prvenstveno dizajniran za mobilne uređaje sa ekranom osetljivim na dodir, kao što su pametni telefoni i tablet uređaji. Korisnički interfejs Androida je zasnovan na direktnoj manipulaciji objektima na ekranu, korišćenjem ulaza u vidu dodira koji odgovaraju pokretima u realnom svetu kao što su prevlačenje, pritiskanje i unos teksta pomoću virtuelne tastature. Varijante Android operativnog sistema se koriste i na igračkim konzolama, digitalnim kamerama, personalnim računarima i drugim elektronskim uređajima. U odnosu na klasično programiranje, programerima je na početku problem da shvate da nema klasičnih konstruktora i inicijalizacija nad androidovim klasama, nego da treba da se prati životni ciklus sistema. Kako aplikacija ima životni ciklus tako i svaka aktivnost ima svoj životni ciklus, od pokretanja aplikacije pa sve do zaustavljanja. Aktivnost predstavlja jednu stranicu koju korisnik vidi i ona spada u jednu od četiri komponente androida. Ostale komponente Androida su servisi, dobavljači sadržaja i prijemnici poruka. Za svaku fazu životnoh ciklusa definisana je odgovarajuća metoda koje su redom nazvane onCreate(), onStart(), onResume(), onPause(), onStop(), onDestory(), onRestart() i pozivaju se u različitim životnim fazama aplikacije. Kao što je već pomenuto, ovakav način programiranja ne liči na do sada viđene i samim tim predstavlja prepreku koja se treba prevazići prilikom inicijalnog razvoja aplikacije. Ulazak u samu materiju i način njegovog funkcionisanja predstavlja poduhvat sam za sebe, pa su programeri mobilnih aplikacija dosta traženi, jer su pored poznavanja samog sistema, za sam razvoj potrebni i uređaji koji koštaju, te nisu svi u mogućnosti da programiraju na njima.

3. KOTLIN

Kotlin [2] je objektno orijentisani programski jezik nastao kao želja za unapređenjem programskog jezika Java [3]. Konkretno Android se na mobilnim telefonima pokreće pomoću Java virtualne mašine, tako što se kreira virtuelna mašina na telefonu, na koju će biti spuštena Android aplikacija. Kotlin programski jezik je razvijen od strane JetBrains kompanije koja stoji iza mnogih alata za pisanje programskog koda i zasniva se na otvorenom kodu.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kupusinac.

Sintaksa koda nije kompatibilna sa Javom, ali Kotlin ima mogućnost komunikacije sa Javom, odnosno mogu se koristiti u istom Java paketu. Android studio od verzije 3.0 zvanično podržava Kotlin kao jezik za izradu Android aplikacija što je uveliko doprinelo popularnosti isog. Kotlin stavlja naglasak na interoperabilnost, sigurnost, jasnoću, lakšu kontrolu pozadinskih procesa i kao takav predstavlja "nadogradnju" Jave. Sledi detaljniji opis neke od najbitnijih funkcionalnosti Kotlina. Funkcija proširenja nam omogućuje da dodatno proširimo već postojeću metodu (ili klasu) bez nasleđivanja. Funkcija visokog nivoa je ona funkcija koja prima funkciju kao parametar ili vraća funkciju. Isto tako, postoje i funkcije prvog reda koje primaju i vraćaju sve vrste parametara osim funkcije. U Kotlinu se null vrednost može lako izbeći upotrebom ? operatora koji prvo proverava da li postoji vrednost. Ako vrednost postoji, program nastavlja da se izvršava, ako ne postoji kod se ne izvršava i ne dolazi do pucanja. Kotlin dopušta imenovanje argumenata koje nam omogućava menjanje redosleda argumenata u funkciji. U Kolinu postoje dve vrste literala: stringovi koji mogu da izbegnu neke karatkere i stringovi koji mogu da sadrže znak za novu linuju. Ovo je u suštini isto kao u Javi, a druga vrsta literala je kada su stringovi sa tri znaka navodnika: """". Unutar para ovih znakova string može da sadrži bilo koji karakter i bezbroj linija. Ovo je prednost u odnosu na javu, koja nema ovu mogućnost. Ponekad nam treba objekat za male izmene neke klase, bez eksplicitne deklaracije nove klase, odnosno singleton. Java ovaj problem rešava pomoću anonimnih unutrašnjih klasa, dok Kotlin realizuje ovaj problem uz pomoću objeknih izraza i deklaracija objekata. Ključna reč za kreiranje singleton klase je object. Opseg se formira uz pomoć rangeTo funkcije koja sadrži operator .. koji se koristi u kombinaciji sa operatorom in. Opseg je difinisan za bilo koji uporediv tip. Za razliku od Jave u Kotlinu klase nemaju statičke metode. U većini slučajeva jednostavno se koriste funkcije na nivou paketa. Ako je potrebno napisati funkciju koja se poziva bez instanciranja određene klase, i potreban je pristup neki poljima unutar te klase, može da se napiše kao član unutar Companion object-a. Još malo približnije, ako se deklariše Companion object unutar jedne klase, mogu da se pozivaju sve metode unutar ovog objekta s istom sintaksom pozivanja statičkih metoda u Javi i C#, koristeći samo ime klase kao kvalifikator. Često se kreiraju klase čija je glavna namena držanje podataka. U takvoj klasi neke standardne fukcionalnosti i korisne funkcije često se mehanički mogu izvesti iz podataka. U Kotlinu ove klase nazivaju se Data class i označavaju se kao podaci. Asinhrono programiranje je sve više potrebno i može se reći da je novo. Bez obzira da li kreiramo server, desktop ili mobilnu aplikaciju važno je da se pruži iskustvo koje nije samo tečnost programa već i skalabilnost kada je potrebno. Postoji mnogo pristupa ovom problemu, a Kotlin ima veoma fleksibilnu podršku za to - korutine. Korutine takođe pružaju mogućnost konkurentnog programiranja. Nema povratnih funkcija, osluškivača ili nešto drugo. Korutina se može zamisliti kao lagana nit. Kao i niti, korutine mogu da se izvršavaju paralelno, mogu čekati jedna drugu i komunicirati. Najveća razlika je u tome što su korutine jeftine, gotovo besplatne, u pogledu preformansi u odnosu na prave niti. Može da se stvori na hiljade korutina i da ne bude toliko zahtevno koliko hiljadu niti može biti ozbiljan izazov za savremenu mašinu.

4. KOMPONENTE SISTEMA

Prilikom razvoja aplikacije korišćene su različite komponente sistema kako bi se poboljšale neke od funkcionalnosti, kao i omogućio brži i lakši razvoj.

4.1. Firebase

Firebase [11] je mobilna i Web platforma koja snabdeva programere sa mnoštvom alata i servisa kako bi im pomogla u razvijanju kvalitetne aplikacije. Nudi nekoliko funkcionalnosti koje su sem analitike koja je odmah dostupna po konektovanju sa projektom, podeljene na tri celine: razvoj, rast i dobit. U ta tri dela su redom smeštene funkcionalnosti prvo koje su nam potrebne uglavnom u fazi razvoja, zatim koje su nam potrebne u fazi rasta aplikacije i funkcionalnosti vezane za marketning. U projektu je korišćena Firebase baza podataka, (*Firebase Realtime Database je Cloud*) podatke čuva u JSON obliku i sinhronizuju se u realnom vremenu sa svim prijavljenim klijentima (korisnicima). Čak i ako napravimo aplikaciju koja može da radi na više platformi (iOS, Android i Web) svi klijenti dobijaju ažurirane podatke [4]. Njene ključne mogućnosti su da u realnom vremenu ne koriste se standardni upiti za podatke, Realtime Database je dizajnirana tako da stalno sinhronizuje podatke sa korisnicima. Svaki put kada se neki podatak u bazi izmeni, svaki konektovan uređaj dobija podatak unutar milisekunde vremena. Zbog toga je Firebase i pogodan za aplikacije koje simuliraju dopisivanje u realnom vremenu i oslanjaju se na žive podatke.

4.2. Ubrizgavanje zavisnosti

Ubrizgavanje zavisnosti spada u jedan od dizajn šablon. Osnova šablona kaže da ukoliko je klasi A potrebna klasa B da završi neki zadatak, može da se kaže kao da je to zavisnost B od A, odnosno A zavisi od B. Čitava poenta sistema jeste da se načini što raslojenijim kako bi bio robusniji i time olakšale buduće izmene istog. Obzirom da se kod mnogo češće čita i prepravlja nego što se piše ili kreće iz početka, potrebno ga je izdeliti u module ili celine koje su maksimalno nezavisne jedne od drugih. Tako se i u ovom radu odlučilo da bude implementiran sličan princip, time je napravljena raslojenost sistema o kojoj će biti više reči u odeljku slojevi arhitekture.

4.3. Gradle sistem automatizacije

Android Studio koristi Gradle [5] sistem za izgradnju, objavljivanje i testiranje aplikacije koji ima mogućnost upravljanja zavisnostima. Gradle je jedan od najpoznatijih sistema za automatizaciju te ga koriste mnogi softverski projekti. Android studio koristi dodatak koji se koristi za komunikaciju sa Gradle sistemom ali nije potpuno zavistan sa Android Studio-m, što znači da se može izgraditi Android aplikacija pomoću komandne linije i bez Android Studia. Isto tako, Gradle je baziran na Java virtualnoj mašini (Java Virtual Machine, JVM [12]), čime je omogućeno pisanje vlastitih skripti u Kotlin programskom jeziku. Android aplikacija sastoji se od dve Gradle datoteke (build.gradle) od kojih se jedna odnosi na aplikacijski modul, a druga na celokupni projekt koja ima zajednička svojstva za sve module. U Gradle datoteci koja se odnosi na modul navedene su neke početne postavke za projekt (aplikacijska oznaka, minimalna verzija SDK [13]

Android paketa, verzija koda itd.), postavke za potpisivanje Android aplikacije kojom možemo ubrzati celi proces te postavke za produkcijsku verziju aplikacije i verziju aplikacije dok je aplikacija u izradi.

5. RAZVOJ APLIKACIJE ZA TRENUTNU RAZMENU PORUKA

U ovom odeljku će biti više reči o funkcionalnosti same aplikacije, detaljnijim opisima samog sistema. Posvetiće se pažnja organizaciji projekta, glavnim svojstvima i karakteristikama aplikacije i arhitekturi koja predstavlja osnovu čitave aplikacije, kao i načinu testiranja istog.

5.1. Pregled i organizacija projekta

Android projekat je generalno organizovan po paketima koji su smešteni unutar vrhovnog paketa koji po običaju ima naziv kao i sama aplikacija. Jedine stvari koje se ne nalaze u njemu, jesu upravo fajlovi i skripte koje služe za pokretanje aplikacije, kao i uvezivanje eksternih zavisnosti poput biblioteka i drugih modula. Taj odvojeni deo se sveobuhvatno nalazi u Gradle Scripts vrhovnom fajlu, gde se nalaze svi Gradle potrebni skripti.

5.2. Arhitektura

Jedan od bitnih faktora prilikom kreiranja projekta jeste arhitektura. Sve više pažnje i značaja daje se arhitekturi. Ona razdvaja komponente i čini projekat modularnijim, lakšim za testiranje i kod lakšim za održavanje. Model-View-ViewModel (MVVM) [6] arhitektura, odnosno šablon je nastala od dobro poznate kompanije Microsoft. MVVM se sve više koristi u izradi Android aplikacija i primarno služi da bi razdvojila prezentacioni sloj od same logike. Ukoliko ne implementiramo okvavku ili sličnu arhitekturu u projekat, sva logika se nalazi u pogledu (View). Na ovaj način kod postaje neprikladan za pisanje testova i nije sklon lakim promenama. Idealna MVVM arhitektura je postignuta kada se može menjati korisnički interfejs, a da logika ostane nepromenjena. Model se odnosi na model domena koji predstavlja stanje sadžaja ili na sloj za pristup podacima koji predstavlja sadržaj. View predstavlja ono što korisnik vidi na ekranu. Prikazuje korisniku podatke koje dobavlja iz modela i stupa u interakciju sa korisnikom pomoću raznih akcija. ViewModel predstavlja apstrakciju modela koji sadrži javna svojstva i komande. Vezivanje podataka omogućuje automatizaciju komunikacije izmedju View-a i ove komponente. Ovde je opisano stanje podataka koji stižu iz Model komponente preko UseCase komponente. UseCase predstavlja sponu izmedju View-a [14] i Model-a i unutar ovog dela arhitekture smešta se biznis logika aplikacije. Slojevi donose razdvajanje korisničkog interfejsa od same logike aplikacije, što omogućava aplikaciji modularnost, testabilnost i održivost koda. Unutar arhitekture razlikujemo tri sloja: 1) Prezentacioni sloj, 2) Domenski sloj, 3) Sloj podataka. Prezentacionom sloju pripadaju delovi pogleda (Fragment-s [16]) i aktivnosti (Activity [15]). Može slobodno da se kaže da su oni "glupi". Izvršuju naredbe i javljaju o događaima koji su se desili od strane korisnika. Prezentacioni sloj vezan je za domenski sloj, koji u sebi sadrži biznis logiku slučajeva korišćenja (UseCase). Skladište podataka (Repository) je deo arhitekture korišćen za obradu podataka. Podaci mogu da se dobiju iz više razlicitih izvora. Podaci o konverzacijama i korisnicima se čuvaju na Cloud Firebase-u, korisnička podešavanja čuvana su u deljene resurse (SharedPreferences [17]). Domain sloj koji rukuje diskretnim delovima poslovne logike aplikacije. Ovo nam omogućuje da centralizujemo biznis logiku i da promena logike bude na jednom mestu. Jedan izvor podataka je implementiran tako da pormene nad bazom podataka obaveste sve one koji je osluškuju da se desila promena. Na ovaj način baza podataka nam služi kao jedini izvor podataka, a drugi delovi joj pristupaju preko skladišta podatka. Bez obzira da li koristimo drugu aplikaciju za dovlačenje podataka, Firebase Database ili neki drugi servis, baza podataka bi trebala da bude jedini izvor.

5.3. Model podataka

Model podataka koji je korišćen dovoljan je za osnovnu funkcionalnost čet aplikacije. Ukoliko bi se aplikacija dalje razvijala sa novim fukcionalnostima neophodno je i proširivanje modela podataka. Model kontakta predstavlja jedinstveno identifikovanog korisnika aplikacije koji se uspešno registrovao na sitem i biće prikazan u listi svih mogućih kontakata u samoj aplikaciji. Konverzacija poredstavlja skladište svih poruka između jednog ili više kontakata, odnosno korisnika sistema, kao i neke od opisnih informacija poput da li korisnik trenutno kuca na konverzaciju, ima i nepročitanih poruka i koliki im je broj i tome sličnih stvari. Sam model čet poruke predstavlja konkretan sadržaj jedne poruke sa svojim tipom, datumom kada je poslata, sadržajem, statusom (primljena, pročitana, neuspelo slanje), pripada tačno jednoj konverzaciji kao i jednom korisniku.

5.4. Opis i funkcionalnosti aplikacije

Aplikacija omogućava ćaskanje sa drugim korisnicima ove aplikacije. Na početnoj strani nalazi se lista konverzacija koje je korisnik ili drugi učesnik započeo i lista korisnika koji koriste aplikaciju. Svaka konverzacija ima ime. Unutar svake konverzacije nalazi se lista poruka koje koje korisnik može da pregleda. U svakom trenutku korisnik može da pošalje novu poruku, a sama poruka može biti različitog tipa, kao na primer običan tekst, slika, lokacija. Svaka poruka ima datum kada je kreirana i status da li je poslata, dostavljena ili pregledana. Korisnik može da započne konverzaciju sa drugim korisnicima aplikacije, pored toga ima mogućnost brze pretrage konverzacija i korisnika. Prijava na sistem je neophodna za aplikaciju zbog identifikacije korisnika. Za prijavu na sistem korišten je Google Sign-In [7]. Ovo nudi bezbedan sistem za potvrđivanje identiteta i smanjuje teret prilikom prijavljivanja korisnika. Omogućuje da se korisnik prijavi sa svojim Google nalogom, isti nalog koji korisi za poštu i druge usluge. Ako korisnik koristi postojeći nalog na Android uređaju, sistem će da ponudi prijavu sa već postojećim nalogom ili pak omogućiti prijavu pomoću novog. Svaki korisnik mora da poseduje Google nalog da bi imao mogućnost prijave na aplikaciju. Za komunikaciju sa udaljenim servisom korišćena je Retrofit biblioteka. Retrofit je REST (Representational state transfer) [19] klijent gradjen nad OkHttp [18] klijentom dizajniran za Android. Koristi anotacije kako bi opisao različite HTTP [20] zahteve, pruža mogućnosti kao što su dinamički parametri unutar URL-a, parametre upita, prilagođena zaglavlja, skidanje i učitavanje datotetka, pravljenje lažnih odgovora (Mocking response). Za izvršavanje Retrofit zahteva potreban je model klasa koja koristi serijalizaciju odgovora, interfejs u kojem je definisan tip HTTP zahteva i Retrofit instanca koja se koristi za izvršavanje HTTP zahteva difinisanog u interfejsu. Svaki

interfejs predstavlja kolekciju različitih Application programming interface (API) poziva, svaka metoda unutar interfejsa koja želi da se koristi kao HTTP zahtev mora da sadrži anotaciju zahteva koji obavlja. Parcijalno učitavanje podataka je odrađeno uz pomoć Paging biblioteke. Ona nudi mogućnost integracije sa RecyclerView-om [8], komponentom androida, koja se i obično koristi za velike liste u kombinaciji sa živim podacima (LiveData). Parcijalno učitavanje može da se implementira na sledeće načine: 1) Samo lokalana baza, 2) lokalna baza i internet, 3) samo server.

5.5. Testiranje

Radi potrebe aplikacije, odrađena su nasumična testiranja koja su ili bila ručno rađena od strane programera ili uz pomoć Monkey Test [21] programa koji nakon podešavanja određenih parametara nasumično prolazi kroz aplikaciju i klikće po ekranu. Pored toga, rađeni su unit testovi, kako bi se proverila konzistentnost prethodnih verzija aplikacije prilikom verzionisanja, unapređivanja i menjanja. Poslednji testovi koji su bili rađeni jesu User Inteface (UI) testovi, kojima se utvrđivali da su određeni pogledi, liste ili pak animacije bile odrađene i prikazane

prilikom različitih stanja aplikacije. Oni su odrađeni uz pomoć biblioteke Espresso-a [9], koja ima mogućnost snimanja akcija korsnika, tako da omogućava reprodukciju određenih slučajeva korišćenja a samim tim i proveru konzistentnosti iste.

6. ZAKLJUČAK

Svrha rada bila je razvijanje aplikacije za ćaskanje uz pomoć najmodernijih tehnologija i osigurati kvalitet razvijene programske podrške. Primećeno je da je bitno obratiti pažnju na arhitekturu programskog rešenja, kako bi se u daljim koracima na brži i lakši način aplikacija mogla graditi i menjati. Korišćenjem najnovijih alata i biblioteka pomoglo je oko implementacije sistema što je svakako preporuka za svakog programera koji bude imao sličnu problematiku. Jedna od mana koja se protezala tokom postavljanja aplikacije jeste dodatno vreme koje je potrebno utrošiti prilikom izrade osnove arhitekture, pogotovo ako programeri nisu upozati sa tehnikama koje se koriste. Ali kao što je već objašnjeno u radu kompletno vreme koje je dodatno uloženo tokom postavljanja projekta i pisanja koda po nekim pravilima uglavnom se isplati već prvom prilikom kada nešto treba menjati u projektu. Upravo iz razloga što će takav kod biti neopisivo lakše održavati nego kod koji je pisan bez jasno definisanih pravila. Kako Google još od kada je kupio Android nastoji da što više olakša posao programerima na ovoj platformi, tako kontinuirano unapređuje biblioteke koje su u njegovom vlasništvu, a koje se koriste za razvoj Android aplikacija. Firebase platforma još od svog nastanka 2011. godine se pokazala kao platforma u koju vredi investirati i koja ima budućnost. Firebase pre svega ubrzava razvoj aplikacije, a i da je to postao Google-ov glavni cilj u održavanju, svedoči činjenica da se sve više aplikacija i softvera opredeljuje za korišćenje njihovih usluga. Postoji mnoštvo servisa iz Firebase-a koji posao od možda jedan ili dva dana kodiranja (prolaske kroz faze razvoja), mogu da obave u pet minuta. Aplikacije za ćaskanje su jedan od prvih primera korišćenja ove platforme, još pod imenom Envolve [10]. Što se tiče praktičnih utisaka, tokom pisanja aplikacije naišlo se na dosta dobru dokumentaciju, čime je dodatno olakšan razvoj. Na osnovu sprovedenog izstraživanja uz oslonac na novo naučene tehnologije i servise, može se zaključiti da bi poznavanje ovih servisa i tehnologija bilo korisno svakom Android programeru i znatno mu olakšalo posao.

Kroz rad se moglo primetiti da su korišćene najnovije tehnologije, ali i ne da su bile jedine prilikom implementacije softverskog rešenja. Šabloni i principi koji su korišćeni su predstavljeni kako bi programeru olakšali proces implementacije softverskog rešenja, ali takođe i ovaj deo može biti zamenjen sa odgovarajućom alternativom. Tokom izrade softverskog rešenja arhitekture, korišćeni alati su se u tom trenutku smatrali za odgovarajućim. Da li je ovo najbolje rešenje? Odgovor na ovo pitanje je teško definisati, tako da se ostavlja čitaocu da sam nađe najrazumnije rešenje za svoj problem uz primer i teorijsku osnovu datu kroz ovaj rad.

7. LITERATURA

[1] Android, official website,
https://www.android.com/
[2] Kotlin, Official documentation,
https://kotlinlang.org/docs/reference/android-overview.html
[3] Java, The Java Language,
https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf
[4] Firebase Realtime Database, official documentation,
https://firebase.google.com/docs/database/
[5] Gradle build system, official website,
https://gradle.org/
[7] Google Sign-in, android developers,
https://developers.google.com/identity/sign-in/android/start-integrating
[8] J. Sonmez, Software Testing Basics, Usersnap,
https://usersnap.com/blog/software-testing-basics/
[9] R. Ajesh, Espresso Basics, Medium,
https://medium.com/mindorks/android-testing-part-1-espresso-basics-7219b86c862b
[10] Envolve
https://www.envolve.com/
[11] Firebase library, Official documentation,
https://firebase.google.com/docs/android/setup
[12] Java virtual machine, JavaWorld,
https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html
[13] Software development kit, SDK, Wikipedia
https://en.wikipedia.org/wiki/Software_development_kit
[14] View, Android Developers
https://developer.android.com/reference/android/view/View
[15] Activity, Android Developers
https://developer.android.com/reference/android/app/Activity
[16] Fragment, Android Developers
https://developer.android.com/guide/components/fragments
[17] SharedPreferences, Android Developers
https://developer.android.com/reference/android/content/SharedPreferences
[18] OkHttp, official github repository
https://github.com/square/okhttp
[19] Representational State Transfer, REST, codecademy
https://www.codecademy.com/articles/what-is-rest
[20] Hypertext Transfer Protocol, HTTP, Wikipedia
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[21] Monckey Testing, Wikipedia
https://en.wikipedia.org/wiki/Monkey_testing

KRATKA BIOGRAFIJA:

Nikola Škrbić rođen je 1994. godine u Novom Sadu. Završio je srednju školu Gimnazija "20. oktobar" u Bačkoj Palanci 2013. godine. Fakultet tehničih nauka u Novom Sadu je upisao 2013. godine. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom.