Android aplikacija za online trgovinu video igara

Ivančok, Deni

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH

TEHNOLOGIJA OSIJEK

Sveučilišni studij

IZRADA ANDROID APLIKACIJE-ONLINE TRGOVINA VIDEO IGARA

Završni rad

Deni Ivančok

Osijek,2022.

Sadržaj

1.UVOD

Cilj ovog završnog rada je omogućiti korištenje online trgovine za kupovinu video igara u obliku Android aplikacije koja implementira, te omogućuje sve potrebne funkcije košarice za online kupovinu. U izradi aplikacije korišten je Android Studio kao razvojno okruženje, programski kod pisan u Kotlin programskom jeziku te je korištena Google-ova Firebase platforma za izradu mobilnih i web aplikacija. Aplikacija omogućava korisniku izbor izrade korisničkog profila te nakon registracije i logiranja pruža odabir igara iz mnoštva kategorija. Proizvodi se klikom na njih dodaju u virtualnu košaricu gdje ostaju spremljeni za svakog pojedinog korisnika sve dok sam korisnik ne odluči dovršiti kupovinu ili ukloniti proizvod iz košarice. Virtualna košarica također pruža povećanje i smanjenje količine istoimenih proizvoda te funkciju sumiranja cijena svih proizvoda te ju sprema kao ukupnu cijenu i prikazuje u aktivnosti same košarice.

U drugom poglavljem su sadržane osnovne informacije te prikazi sličnih aplikacija dostupnih na Trgovini Play. U trećem i četvrtom poglavlju rada opisuje se rad u aplikaciji te alati korišteni u izradi aplikacije zajedno s izlistanim kodom. Nakon toga je prikazan sam rad aplikacije.

1.1 Zadatak završnog rada

Kratko opisati način funkcioniranja web trgovine. Izraditi i opisati postupak izrade Android aplikacije koja će se koristiti kao online trgovina video igara. Predvidjeti mogućnost registracije novih korisnika u aplikaciji kao i pohranu korisničkih podataka na udaljenom poslužitelju baze podataka. Obaviti ručno testiranje rada izrađene aplikacije i opisati postupak testiranja.

2. ONLINE KUPOVINA

Ovim poglavljem opisana je online kupovina, njene prednosti te nedostaci. Navedeno je nekoliko primjeraka postojećih online trgovina te su analizirane implementacije njihovih virtualnih košarica. U poglavlju analiza sličnih implementacija, navedene su sve već spomenute online trgovine i uspoređena je implementacija njihovih virtualnih košarica.

2.1 Općenito o online kupovini

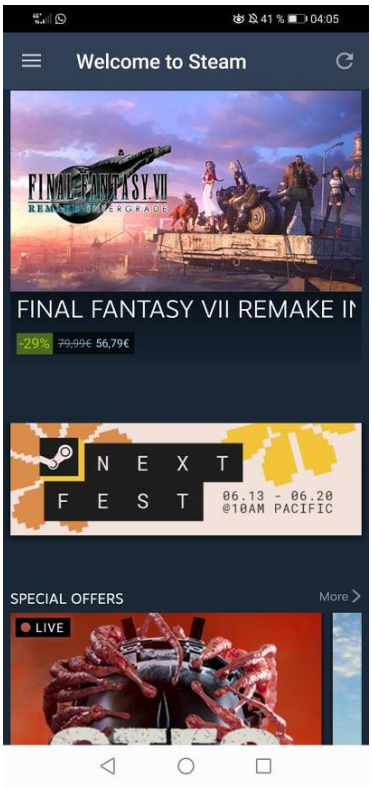
Kupovina proizvoda putem interneta naziva se online kupovina. S vremenom se počela koristiti gotovo istom učestalošću kao i fizička. U online kupovini kupac odlazi na internet kako bi potražio proizvode na web stranici prodavatelja i odabrao proizvod za kupnju. Internetska kupovina pruža mnoge pogodnosti pri kupovini. Kupci mogu kupovati s gotovo bilo kojeg mjesta na kojem se trenutno nalaze sve dok imaju uređaj koji je to u mogućnosti napraviti. Kupovina putem interneta olakšala je kupnju jer nema čekanja tako da je moguće obaviti kupovinu u kratkom vremenu i u bilo koje doba dana. Online kupovina pruža detaljne informacije o proizvodu te mogućnost ostvarivanja popusta i nižih cijena. Također nije potrebno suočavati se s gužvama. Ponekad trgovci mogu korištenjem svojih komunikacijskih vještina uvjeriti kupca na kupnju određenog proizvoda. Kao rezultat toga, ponekad kupac kupuje one stvari koje mu zapravo nisu potrebne dok online kupovina ne posjeduje takvu direktnu sugestivnost poput prodavača uživo. Nadalje, olakšava traženje proizvoda unosom nekih ključnih riječi ili korištenjem tražilica. Većina potrošača čita online recenzije proizvoda kako saznali osnovne informacije koje su napisali kupci koji su ga već koristili. Online recenzije pomažu u dobivanju povratnih informacija o proizvodu za koji su kupci zainteresirani. Online kupovina također bilježi značajan porast u zadnje dvije godine što je uzrokovano pandemijom Covid-19. [1]

2.2 Slična rješenja mobilnih online trgovina

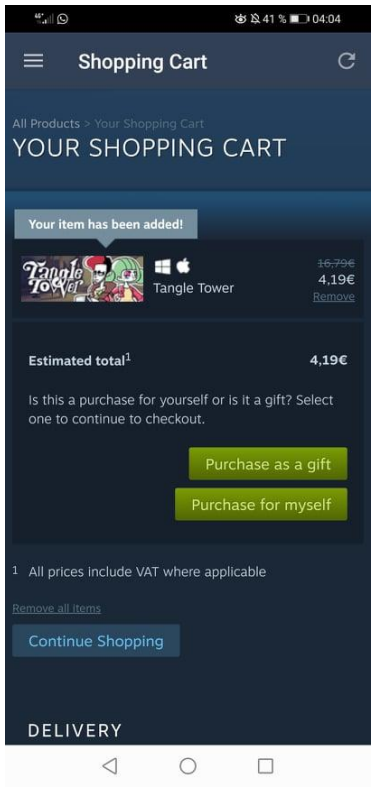
2.2.1 Steam

Steam [2] je platforma za digitalnu distribuciju videoigara koju je kreirala korporacija Valve. To je zajednica u kojoj igrači i programeri igara mogu kupovati i prodavati videoigre na mreži kao što je prikazano na slici 2.1, svim korisnicima je omogućeno pisanje osobnih mišljenja i recenzija za igre koje se nalaze na platformi te omogućuje komunikaciju i zajedničko igranje istih. Steam je jedna od najpopularnijih zbog svojih karakteristika koje privlače korisnike. Značajan primjer za to su njegove usluge korisničke podrške zajedno s alatima i značajkama za pomoć i cjelokupno poboljšanje korisničkog iskustva igranja. Ima preko 47 milijuna aktivnih korisnika dnevno, te djeluje gotovo 16 godina. Kod Steam aplikacije može se vidjeti prilično jednostavna

implementacija košarice s prikazom igre, njene cijene, mogućnošću brisanja stavke iz košarice te mogućnost kupnje stavke kao dar ili za vlastito korištenje (slika 2.2).



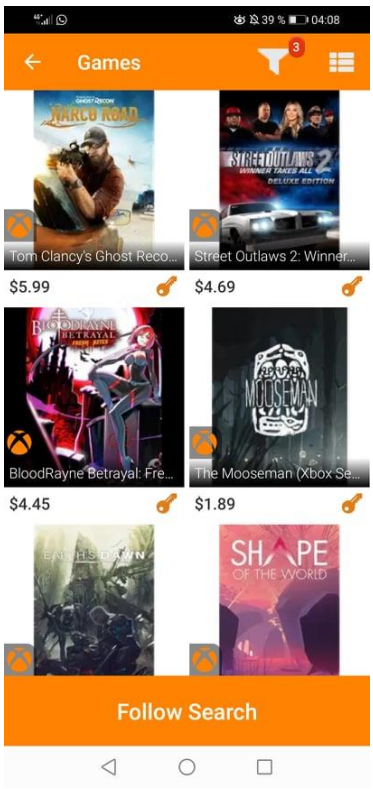
Slika 2.1 Prikaz glavnog izbornika Steam aplikacija



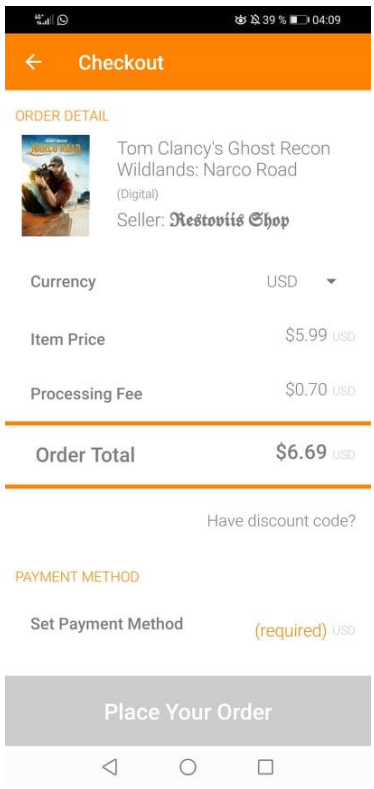
Slika 2.2 Prikaz košarice Steam aplikacije

2.2.2 Gameflip

Gameflip[3] je tehnološka tvrtka usredotočena na inovacije koja stvara komercijalni motor za gaming omogućavajući svim sudionicima poput igrača, kreatora, robnih marki i programera povezivanje te sigurno vođenje trgovine i međusobno dijeljenje prednosti. Gameflip Market omogućuje i prodaju raznih predmeta uključujući NFT-ove (eng. Skraćenica Non-fungible token), to jest digitalne kolekcionarske predmete, predmete u igri, darovne kartice te naravno same igre (prikazano na slici 2.3). Gameflip Marketu vjeruje zajednica od 6 milijuna igrača koji su ostvarili promet od preko 120 milijuna dolara. Izgrađen je na provjerenoj arhitekturi koja je pružila sigurnost i skalabilnost za milijune digitalnih transakcija robe i imovine. Košarica Gameflip aplikacije također ima vrlo jednostavnu implementaciju s prikazom igre i njene cijene te tipkom za potvrdbu narudbe (slika 2.4) .



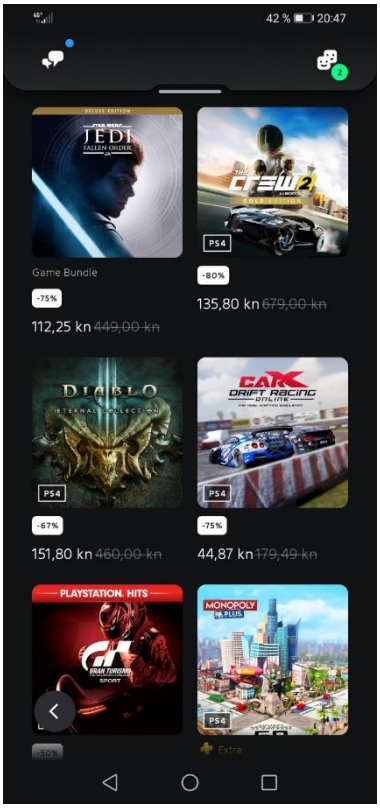
Slika 2.3 Prikaz Izbornika Gameflip aplikacije



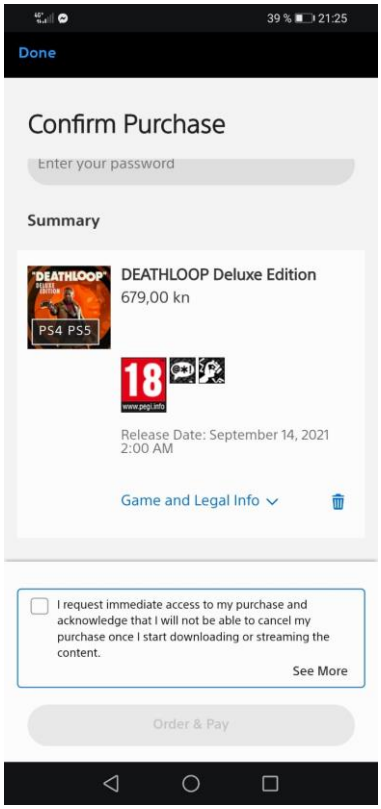
Slika 2.4 Prikaz košarice Gameflip aplikacije

2.2.3 Playstation Network

PlayStation Network je usluga za online igranje i distribuciju medijskih sadržaja (slika 2.5). Sony Corporation izvorno je stvorio PSN kako bi podržao svoju PlayStation 3 igraću konzolu. Tvrtka je proširila uslugu tijekom godina kako bi podržala druge Sony uređaje, uz to i prijenos glazbe i video sadržaja[4]. PlayStation Network je u vlasništvu i pod upravom Sony Network Entertainment International. S vremenom postaju nalik moblinoj verziji Steam aplikacije. Također omogućuje komunikaciju, zajedničko igranje igara te igračima otključavanje i dijeljenje trofeja postignutim u igrama s drugim korisnicima. Trofeji su ciljevi koje je moguće ispuniti u igrama, kao što je dovršavanje razine bez umiranja ili prikupljanje svih predmeta unutar igre. Normalni trofeji dolaze u brončanoj, srebrnoj i zlatnoj varijanti ovisno o njihovoj težini. Platinasti trofeji su posebni i otključavaju se tek kada korisnik otključa sve ostale prethodne trofeje za igru. Sve PlayStation online značajke, kao što su Party chat, Remote Play i Share Play, koriste temelj PlayStation Networka za funkcioniranje. Kao rezultat toga, ne postoji web stranica ili posebna usluga koja unificira sve ostale usluge PlayStation Network-a već se radi o složenoj infrastrukturi koja pokreće mnoge usluge. Također ima prilično jednostavnu implementaciju košarice s igrom njenim podacima, cijenom te tipkama za brisanje i potvrde kupnje stavke (slika 2.6).



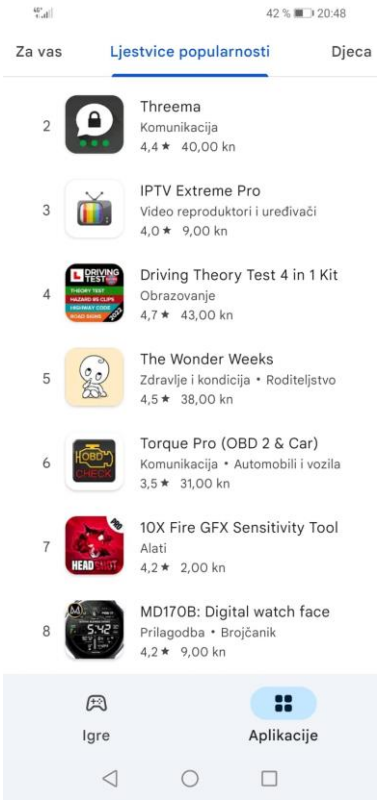
Slika 2.5 Prikaz igara playstation network



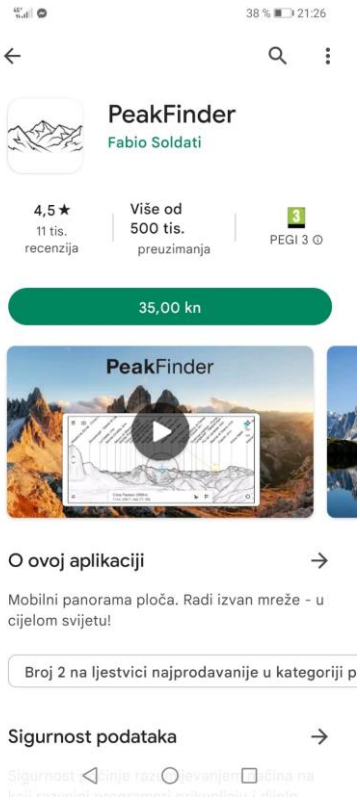
Slika 2.6 Prikaz košarice playstation network-a

2.2.4 Trgovina Play

Google Play je internetska trgovina u kojoj ljudi traže svoje omiljene aplikacije, igre, filmove, TV emisije, knjige i još mnogo toga. Omogućuje 2 milijuna aplikacija i igara milijardama ljudi diljem svijeta, generirajući više od 120 milijardi dolara zarade programerima [5]. Google Play Store je srce Google Play usluge. Omogućuje pristup opsežnoj biblioteci aplikacija i igara od Googlea i bilo kojeg drugog programera kojemu je u cilju postići što veću izloženost što široj publici za svoje Android aplikacije. Trgovina Play sadrži mješavinu besplatnih i plaćenih aplikacija u širokom rasponu kategorija, uključujući društvene medije, zabavu, produktivnost i fotografiju (slika 2.7). Google također dopušta razvojnim programerima i korisnicima korištenje svoje platforme Google Play za upravljanje kupnjom aplikacija, pretplatama i plaćanjima unutar aplikacije. Do sada najjednostavnija implementacija košarice pripada Google Play aplikaciji. Klasična implementacija košarice ne postoji. Sadrži samo prikaz aplikacije, broj preuzimanja, recenzije, informacije te tipku za kupnju s ponuđenom cijenom (slika 2.8). Pritiskom na tipku ponudi se izbornik za način plaćanja te se izvršava kupnja aplikacije.



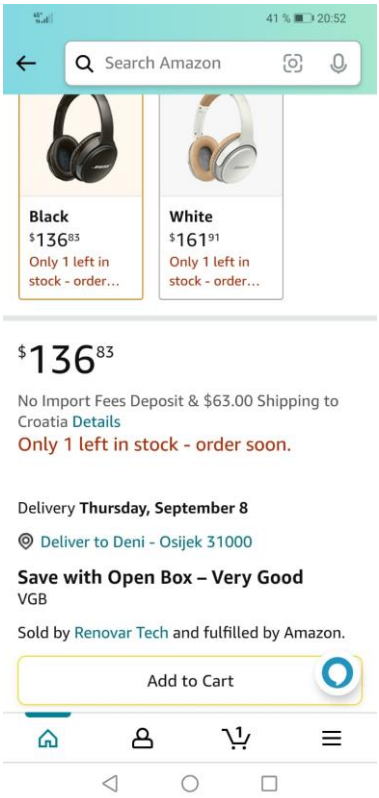
Slika 2.7 Prikaz Izbornika Google Play aplikacije



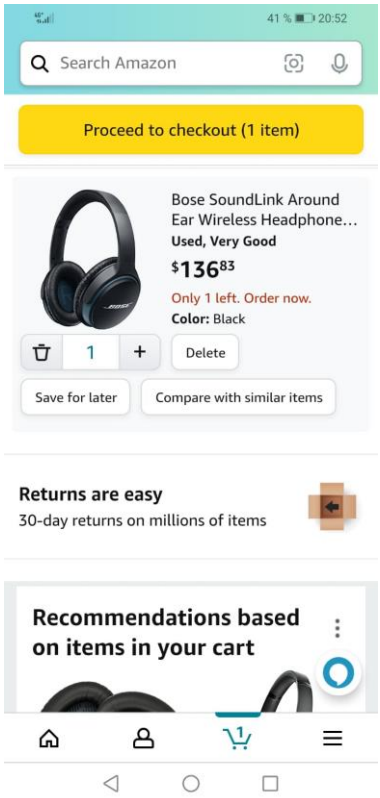
Slika 2.8 Prikaz kupnje aplikacije na Google Play trgovini

2.2.5 Amazon

Amazon je najveći svjetski online trgovac na malo i istaknuti pružatelj usluga u oblaku. Izvorno započet kao tvrtka za online prodaju knjiga, Amazon se pretvorio u internetsko poslovno poduzeće koje je uvelike usmjereno na pružanje usluga e-trgovine, računalstva u oblaku, digitalnog strujanja i usluga umjetne inteligencije (AI) [6]. Tvrtka nudi monumentalni asortiman proizvoda i inventar, omogućujući potrošačima kupnju gotovo svega, uključujući odjeću, kozmetičke potrepštine, gurmansku hranu, nakit, knjige, filmove, elektroniku, potrepštine za kućne ljubimce, namještaj, igračke, vrtni pribor i potrepštine za kućanstvo (slika 2.9). Sa sjedištem u Seattleu, Amazon sadrži pojedinačne web stranice, centre za razvoj softvera, centre za korisničku podršku, podatkovne centre i centre za isporuku po cijelom svijetu. S obzirom da je Amazon najveći svjetski online trgovac to se odražava i na dizajn njihovih stranica i aplikacija za online kupovinu. Aplikacija posjeduje dosta detaljno razrađenu košaricu u odnosu na prošle implementacije. Pruža informacije o stanju i svim postojećim specifikacijama artikla. Posjeduje tipku za povećanje količine, brisanje, spremanje za kasnije i usporedbu s ostalim artiklima (slika 2.10).



Slika 2.9 Prikaz Izbornika Amazon trgovine



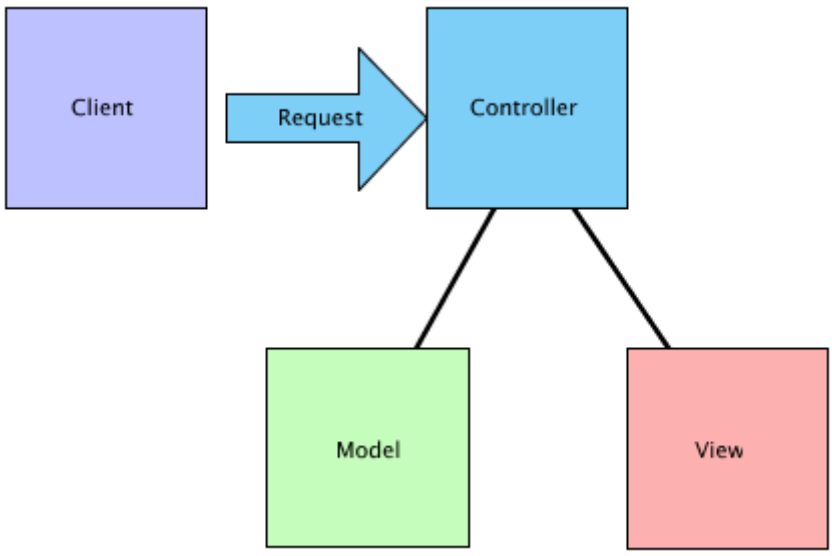
Slika 2.10 Prikaz košarice Amazon trgovine

2.2.6 Analiza sličnih implementacija online trgovina

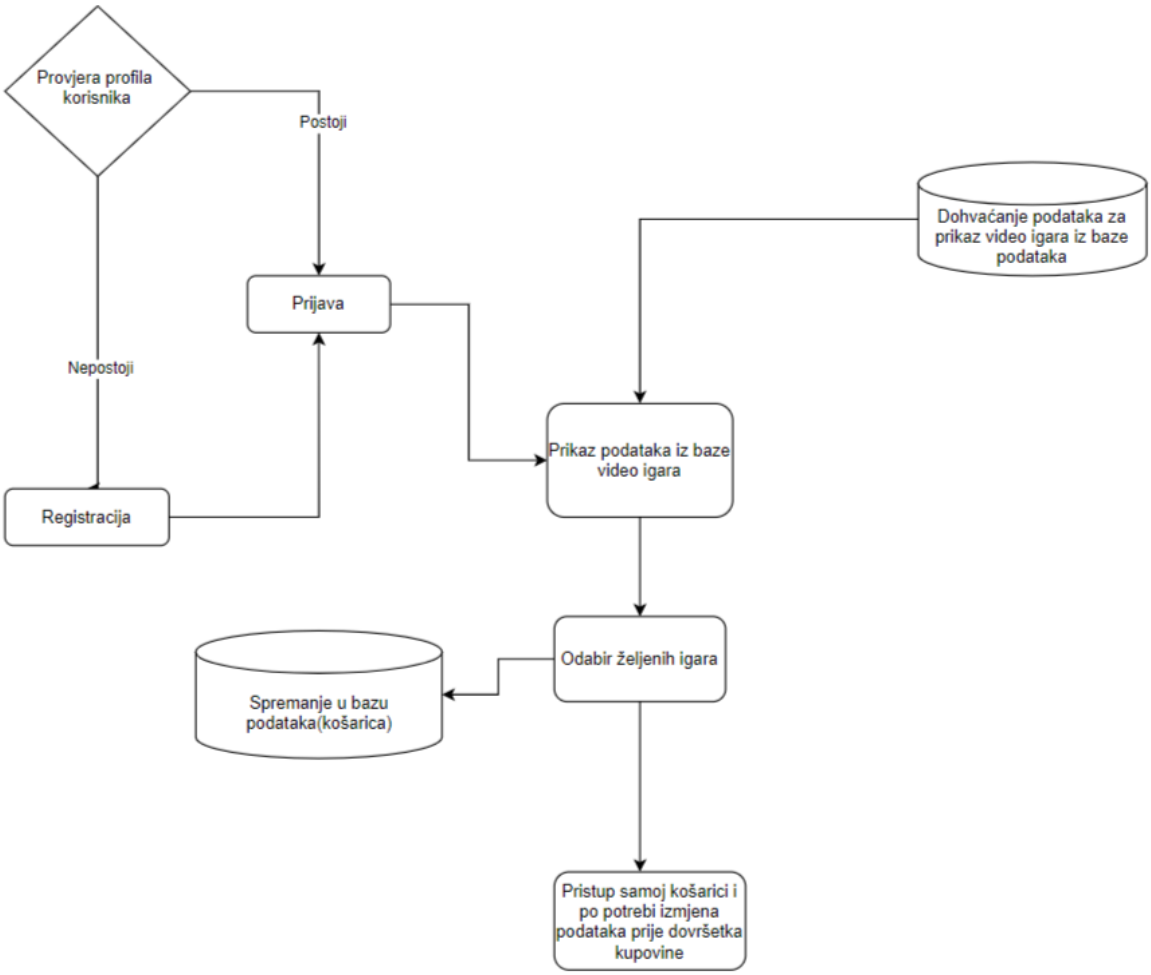
Svi navedeni primjeri prikazuju kako je korisnicima omogućena online kupnja video igara i, po mogućnosti drugih, proizvoda pomoću implementacija virtualnih košarica. Sačinjene su od osnovnih funkcija dodavanja i brisanja određenih stavki iz košarice osim u slučaju *Google Play* trgovine. Prikazuju ukupnu cijenu nastalu sumom svih stavki unutar nje, te pružaju osnovne informacije o igrama pa čak i korisničke recenzije u slučaju *Steam, Amazon i Google play* aplikacije. Daleko najopsežniju implementaciju posjeduje već spomenuta *Amazon* aplikacija.

3.ARHITEKTURA APLIKACIJE

Uvelike se koristi arhitektonski obrazac *Model View Controller* (MVC), kao što je prikazano na slici 3.1, u kojemu se *Activity* i *Fragmenti* koriste kao upravljači i modeli za podatke i njihovu dosljednost. Model predstavlja podatke i znanje unutar sustava. Podaci obično dolaze iz baze podataka međutim nisu ograničeni na nju. Model dohvaća podatake iz baze podataka i može uključivati poslovnu logiku . Pogled je odgovoran za prikaz modela na ekranu. Kontroler povezuje korisnika, model i pogled zajedno, uzimajući zahtjev korisnika, spajajući ga s odgovarajućim modelom i kombinirajući model s odgovarajućim pogledom [7]. Prednosti ovoga obrasca uključuju mogućnost ponovne upotrebe, na primjer ponovno korištenje istog kontrolera što poboljšava mogućnost održavanja jer je lakše pronaći, popraviti i poboljšati stvari i mogućnost testiranja, jer se svaka komponenta može testirati zasebno. Naravno to nosi i određene nedostatke koji dolaze s povećanjem koda, te *Activity* i *Fragmenti* postaju prenatrpani. U tom slučaju pribjegava se *Model View Presenter* (MVP) ili *Model View View Model* (MVVM)[8] obrascima dizajna aplikacije, međutim za rad izrađene aplikacije nije bilo potrebe koristiti gore navedene arhitekture. Slika 3.2 prikazuje dijagram toka [9] kao idejno rješenje vlastite aplikacije.



3.1 Slika Model-Kontroler-Pogled dizajn arhitekture



3.2 Slika Dijagram toka aplikacije

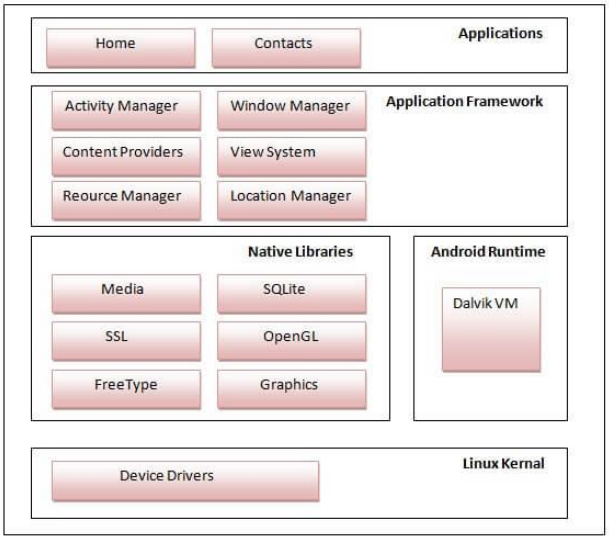
4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

Ovim poglavljem opisane su sve korištene programske tehnologije, jezici i razvojna okruženja korištena za izradu aplikacije. Detaljno je opisan te prikazan kod koji predstavlja sve funkcionalnosti unutar izrađene aplikacije.

4.1 Korištene programske tehnologije, jezici i razvojna okruženja

4.1.1 Mobilni operativni sustav Android

Android je mobilni operativni sustav koji je razvila tvrtka Google te postoji skoro 15 godina. Godine 2007. objavljena je prva verzija operativnog sustava koja se naziva Android 1.0 *Apple Pie*. Temelji se na jezgri Linux operativnog sustava koji je besplatan, multiplatformni sustav softvera otvorenog koda i jedan je od primarnih operativnih sustava mobilnih uređaja odmah uz Apple-ov iOS. Nudi ujedinjen pristup razvoju aplikacija za mobilne uređaje što znači da programeri trebaju razvijati samo za Android, a njihove aplikacije trebale bi imati mogućnost izvoditi se na različitim uređajima koje pokreće Android [10].



Slika 4.1 Arhitektura Android operativnog sustava (preuzeto s https://www.javatpoint.com/android-software-stack)

Aplikacije su najviši sloj Android arhitekture. Unaprijed instalirane aplikacije kao što su *Home*, kontakti, kamera, galerija i ostale, te aplikacije trećih strana preuzete iz trgovine Play kao što su aplikacije za chat, igre bit će instalirane samo na ovom sloju. Rade unutar ART-a (eng. *Androida Runtime)* uz pomoć klasa i usluga koje pruža *Application Framework*. Android *Runtime* okruženje jedan je od najvažnijih dijelova Androida. Sadrži komponente kao što su osnovne knjižnice i Dalvik virtualni stroj (eng. *Dalvik Virtual Machine*). Uglavnom, pruža osnovu za okvir aplikacije i pokreće aplikaciju uz pomoć osnovnih knjižnica. Poput *Java Virtual Machine* (JVM), DVM [11]

je virtualni stroj koji se temelji na registrima i posebno je dizajniran i optimiziran za Android kako bi se osiguralo da uređaj može učinkovito pokretati više instanci. Ovisi o sloju Linux kernela za upravljanje nitima i memorijom niske razine. Osnovne knjižnice omogućuju implementaciju Android aplikacija korištenjem standardnih JAVA ili Kotlin programskih jezika.*Application Framework* pruža nekoliko važnih klasa koje se koriste za izradu Android aplikacije. Pruža generičku apstrakciju za pristup hardveru i također pomaže u upravljanju korisničkim sučeljem s resursima aplikacije. Općenito, pruža usluge uz pomoć kojih se mogu stvoriti određene klase i učiniti te klasu korisnom za kreiranje aplikacija. Uključuje različite vrste upravitelja aktivnosti usluga, upravitelja obavijesti, sustava pregleda, upravitelja paketa koji su korisni za razvoj aplikacije prema zadanim preduvjetima. Knjižnice platforme uključuju različite C/C++ osnovne biblioteke temeljene na Javi kao što su Media, Graphics, Surface Manager, OpenGL kako bi pružile podršku za razvoj Androida. Biblioteka medija pruža podršku za reprodukciju i snimanje audio i video formata. Upravitelj površine odgovoran za upravljanje pristupom podsustavu zaslona. *SGL(*eng. *Scalable Graphics Library)*, *OpenGL(*eng. *Open Graphics Library),* i *API*(eng. *Application Programming Interface*) koriste se za 2D i 3D računalnu grafiku. *SQLite* pruža podršku za rad s bazom podataka, a FreeType pruža podršku za fontove. *Web-Kit* je pogon otvorenog koda razvijen od strane Apple-a te pruža sve funkcije za prikaz web sadržaja i pojednostavljenje učitavanja stranica. SSL (eng. *Secure Sockets Layer*) je sigurnosna tehnologija za uspostavljanje šifrirane veze između web poslužitelja i web preglednika. Linux jezgra je srce Android arhitekture. Upravlja svim dostupnim upravljačkim programima kao što su upravljački programi za zaslon, upravljački programi kamere, Bluetooth upravljački programi, audio upravljački programi, upravljački programi za memoriju i ostali koji su potrebni tijekom rada. Linux kernel će osigurati sloj apstrakcije između hardvera uređaja i ostalih komponenti Android arhitekture [12]. Odgovoran je za upravljanje memorijom, napajanjem, uređajima.

4.1.2. Programski jezik Kotlin

Kotlin je programski jezik opće namjene, besplatan, otvorenog koda, statički tipiziran "pragmatični" programski jezik prvobitno dizajniran za JVM [13] i Android koji kombinira objektno orijentirane i funkcionalne programske značajke te je sažetiji od srodnog programskog jezika Jave [14]. Usredotočen je na interoperabilnost, sigurnost, jasnoću i podršku alatima. U proizvodnji su i verzije Kotlina koje ciljaju na *JavaScript* ES5.1 i izvorni kod koristeći LLVM(eng. Low Level Virtual Machine) za brojne procesore. Kotlin je razvila tvrtka JetBrains tvrtke koja stoji iza IntelliJ IDEA, 2010. godine, a otvoren je kod od 2012. Tim Kotlina trenutno ima više od 90 stalnih članova iz JetBrainsa, a projekt Kotlin na GitHubu ima više od 300 suradnika.

4.1.3. Strukturni jezik XML

XML (eng. *Extensible Markup Language*) je označni jezik sličan HTML (eng. *HyperText Markup Language*), ali bez unaprijed definiranih oznaka za korištenje. Umjesto toga, korisnik definira se vlastite oznake dizajnirane posebno za korisničke potrebe. Ovo je moćan način pohranjivanja podataka u formatu koji se može pohranjivati, pretraživati i dijeliti. Ono što je najvažnije, budući da je temeljni format XML-a standardiziran, ako se dijeli ili prenosi preko sustava ili platformi, bilo lokalno ili preko interneta, primatelj još uvijek može analizirati podatke zahvaljujući standardiziranoj XML sintaksi. Svi elementi sučelja grade se korištenjem hijerarhije *View* i *ViewGroup* objekata, gdje su *View* objekti nešto što korisnik vidi, a *ViewGroup* objekti su objekti koji sadrže *View* objekte.

4.1.4. Integrirano razvojno okruženje Android Studio

Android Studio je službena razvojna okolina za stvaranje Android mobilnih aplikacija. Omogućuje korištenje Jave i Kotlina kao programskih jezika [16]. Sadrži razne alate za razvoj i pisanje koda te emulator na kojemu se može aktivno pokretati aplikacije i bez prisutnih mobilnih Android uređaja te niz ostalih vrlo korisnih značajki.

4.1.5. Firebase platforma

Firebase je *Backend-as-a-Service* (BaaS), omogućuje programerima usmjeravanje pažnje na prednji dio svojih aplikacija i korištenje pozadinske usluge bez njihove izgradnje ili održavanja. Programerima pruža razne alate i usluge koje im pomažu razviti kvalitetne aplikacije, povećati svoju korisničku bazu i zaradu. Izgrađen je na Googleovoj infrastrukturi. Firebase je kategoriziran kao *NoSQL(eng.* Structured Query Language*)* program baze podataka, koji pohranjuje podatke u *JSON* dokumentima. Sadrži razne značajke poput autentifikacije pomoću zaporka, telefonskih brojeva, Googlea, Facebooka, Twittera i još mnogo toga. *Firebase* autentifikacija može se koristiti za ručnu integraciju jedne ili više metoda prijave u aplikaciju. Posjeduje bazu podataka u stvarnom vremenu u kojoj se podaci sinkroniziraju na svim klijentima u stvarnom vremenu i ostaju dostupni čak i kada se aplikacija isključi. *Firebase* Hosting pruža brzi hosting za web aplikaciju; sadržaj se sprema u mreže za isporuku sadržaja diljem svijeta. Aplikacija se može testirati putem Firebase-a na virtualnim i fizičkim uređajima koji se nalaze u Google-ovim podatkovnim centrima te se obavijesti mogu slati bez dodatnog kodiranja [17].

4.2. Programsko rješenje na strani korisnika

4.2.1 Zaslon za Registraciju korisnika

Pri prvom pokretanju aplikacije otvara se zaslon za prijavu korisnika. Kako bi se korisnik mogao prijaviti potrebno se registrirati klikom na *sign up* tipku koja će ga odvesti na aktivnost za registraciju gdje će biti potrebno unjeti e-mail i šifru koju će koristiti za pristup aplikaciji. Ovo je ostvareno kodom u *RegisterActivity*.

```
findViewById<Button>(R.id.signup_button).setOnClickListener {  it: View!
    when {
        TextUtils.isEmpty((email.text).toString()) -> {
            Toast.makeText(
                context: this@RegisterActivity,
                text: "Molim vas unesite vašu e-mail adresu.",
                Toast.LENGTH_SHORT
            ).show()
        }

        TextUtils.isEmpty((pass.text).toString()) -> {
            Toast.makeText(
                context: this@RegisterActivity,
                text: "Molim vas unesite vaš password.",
                Toast.LENGTH_SHORT
            ).show()
```

Programski kod 4.1 provjera upisivanja podataka pri registraciji korisnika

```
rtAdapter.kt × MyGameAdapter.kt × RegisterActivity.kt × CartActivity.kt × HorrorIgre.kt × HeroIgre.kt × FightIgre.kt × LogInActivity.kt ×
            TextUtils.isEmpty((pass.text).toString()) -> {
                Toast.makeText(
                    context: this@RegisterActivity,
                    text: "Molim vas unesite vaš password.",
                    Toast.LENGTH_SHORT
                ).show()
            }
            else -> {
                val email: String = findViewById<EditText>(R.id.reg_email).text.toString()
                val pass: String = findViewById<EditText>(R.id.reg_password).text.toString()

                FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, pass)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            Toast.makeText( context: this@RegisterActivity, text: "Uspješno ste regstirirani.",
                                Toast.LENGTH_SHORT).show()

                            val intent= Intent( packageContext: this@RegisterActivity,LogInActivity::class.java)
                            startActivity(intent)
                            finish()

                        }else{
                            Toast.makeText( context: this,task.exception!!.message.toString(), Toast.LENGTH_SHORT).show()
                        }
                    }
            }
        }
    }
}
```

Programski kod 4.2 Registracija korisnika putem Firebase API

RegisterActivity prvotno provjerava jesu li EditText-ovi za unos e-mail adrese i šifre prazni pomoću *TextUtils.isEmpty()* te zahtjeva njihov unos ako jesu prikazano na slici 4.1. U suprotnom slučaju uzima string unesene e-mail adrese i šifre te predaje *createUserWithEmailAndPassword()* metodi koja je dio FirebaseAuth klase i koja sprema te

korisničke podatke u bazu podataka Firebase-a prikazano na slici 4.2. Na uspješno izvršen zadatak obaviještava korisnika kako je uspješno registriran i vraća na početnu *LoginActivity*.

4.2.2 Zaslon za Prijavu korisnika

Nakon izvršene registracije *LoginActivity* omogućuje prijavu korisnika sa vrlo sličnom implementacijom koda kao kod *RegisterActivity*.

```
val email = findViewById<EditText>(R.id.reg_email)
val pass = findViewById<EditText>(R.id.reg_password)
findViewById<Button>(R.id.button).setOnClickListener {  it: View!
    when {
        TextUtils.isEmpty((email.text).toString()) -> {
            Toast.makeText(
                context: this@LogInActivity,
                text: "Molim vas unesite vašu e-mail adresu.",
                Toast.LENGTH_SHORT
            ).show()
        }

        TextUtils.isEmpty((pass.text).toString()) -> {
            Toast.makeText(
                context: this@LogInActivity,
                text: "Molim vas unesite vaš password.",
                Toast.LENGTH_SHORT
            ).show()
```

Programski kod 4.3 provjera upisivanja podataka pri registraciji

```
val email: String = findViewById<EditText>(R.id.reg_email).text.toString()
val pass: String = findViewById<EditText>(R.id.reg_password).text.toString()
val name:String= findViewById<EditText>(R.id.name).text.toString()
FirebaseAuth.getInstance().signInWithEmailAndPassword(email, pass)
    .addOnCompleteListener { task ->
        if (task.isSuccessful) {
            Toast.makeText(
                context: this@LogInActivity,
                text: "Uspješno ste se ulogirali.",
                Toast.LENGTH_SHORT
            ).show()
            val intent =
                Intent( packageContext: this@LogInActivity, MainActivity::class.java).also{  it: Intent
                    it.putExtra( name: "EXTRA_NAME",name)
                    it.putExtra( name: "EXTRA_EMAIL",email)
                    startActivity(it)
                    finish()
                }
        }else{
            Toast.makeText( context: this,task.exception!!.message.toString(), Toast.LENGTH_SHORT).show()
        }
    }
}
}
}
```

Programski kod 4.4 Prijava korisnika putem Firebase API

LoginActivity bas kao i Register activity provjerava jesu li EditText-ovi za unos e-mail adrese i šifre prazni pomoću *TextUtils.isEmpty()* te zahtjeva njihov unos ako jesu prikazano na slici

4.4. U suprotnom baš kao i *RegisterActivity* uzima string unesene e-mail adrese i šifre no to predaje *signInWithUserWithEmailAndPassword()* metodi koja je također dio FirebaseAuth klase prikazano na slici 4.5. Na uspješno izvršen zadatak obaviještava korisnika kako je uspješno prijavljen i prosljeđuje ga na MainActivity.

4.2.3 Glavni izbornik i RecyclerView

Za implementaciju *RecyclerView*-a potrebne su tri komponente: sučelje koje pruža klikabilnost u programskom kodu adapter koji upravlja podacima koji će se naći unutar *RecyclerView*-a i klasa koja opisuje jedan element *RecyclerView*-a. *RecyclerView* implementira se na način da, nakon nasljeđivanja, prepišu metode koje pruža roditeljska klasa, to su *onCreateViewHolder, onBindViewHolder i getItemCount (slika 4.7)*. Prve dvije navedene metode bave se ispunjavanjem određenog elementa željenim podacima i njegovim prikazom, dok *getItemCount* vraća broj elemenata unutar *RecyclerView*-a. Klasa koja predstavlja jedan element unutar *RecyclerView*-a treba nasljeđivati klasu *RecyclerView.ViewHolder* i ako se želi postići klikabilnost potrebno je implementirati vlastito sučelje *IRecyclerClickListener*. Kako bi se implementiralo sučelje *IRecyclerClickListener,* potrebno je prepisati metodu *onClick*, koja omogućuje definiranje željene radnje prilikom pritiska na jedan element *RecyclerVi*ew-a. Budući da je navedeno sučelje implementirano, kao argument za metodu *setOnClickListener* može se predati referenta varijabla *IRecyclerClickListener sučelja prikazano na slici 4.6*. Unutar *onBindViewHolder-a (slika 4.7)* holder poziva *setClickListener* kojem se predaje objekt sučelja te metoda *onItemClickListener* iz sučelja koja poziva metodu *addToCart* kojoj predaje poziciju unutar liste. Implementacije metoda *OnClickListener, onClick, onCreateViewHolder, onBindViewHolder i getItemCount* su pikazane na slikama ispod.

```
class MyNajnovijeAdapter(
    private val context: Context,
    private val list: List<GameModel>,
): RecyclerView.Adapter<MyNajnovijeAdapter.MyGameViewHolder>() {

    class MyGameViewHolder(itemView: View): RecyclerView.ViewHolder(itemView),
        View.OnClickListener {
        var imageView: ImageView?=null
        var txtName: TextView?=null
        var txtPrice: TextView?=null

        private var clickListener: IRecyclerClickListener?= null

        fun setClickListener(clickListener: IRecyclerClickListener){
            this.clickListener=clickListener
        }

        init {
            imageView=itemView.findViewById(R.id.imageView) as ImageView
            txtName=itemView.findViewById(R.id.txtName) as TextView
            txtPrice=itemView.findViewById(R.id.txtPrice) as TextView
            itemView.setOnClickListener(this)
        }

        override fun onClick(v: View?) {
            clickListener!!.onItemClickListener(v,adapterPosition)
        }
```

Programski kod 4.5 implementacija klikabilnosti RecyclerView-a

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyGameViewHolder {
    return MyGameViewHolder(
        LayoutInflater.from(context).inflate(R.layout.layout_najnovije_item, parent, attachToRoot: false)
    )
}

@SuppressLint( ...value: "SetTextI18n")
override fun onBindViewHolder(holder: MyGameViewHolder, position: Int) {
    Glide.with(context).load(list[position].image).into(holder.imageView!!)
    holder.txtName!!.text= StringBuilder().append(list[position].name)
    holder.txtPrice!!.text=list[position].price

    holder.setClickListener(object: IRecyclerClickListener {
        override fun onItemClickListener(view: View?, position: Int) {
            addToCart(list[position])
        }
    })
}

override fun getItemCount(): Int {
    return list.size
}
```

Programski kod 4.6 implementacija 3 glavne metode implementacije Recyclerview-a

4.2.4 Metoda addtoCart, onDataChange i DataSnapshot

addtoCart (slika 4.8) metoda smještena je unutar dva adaptera *RecyclerView-a* (*MyNajnovijeAdapter* i *MyGameAdapter)* koji su korišteni za predstavljanje lista igara te joj se predaje instanca *GameModela (podac).* Metoda pristupa *Realtime* bazi podataka unutar *Firebase* konzole koja sinkronizira *JSON* podatke. Pod putanju "Cart" smještenu u bazi podataka upisuje jedinstveni uid trenutno prijavljenog korisnika prilikom klika na neki od RecyclerView-ova to je sve navedeno unutar *userCart* vrijednosti. Nadalje *userCart* poziva te u pod putanju uid trenutno prijavljenog korisnika sprema ili ažurira podatke o odabranoj igri. Na ovaj način se realizira spremanje igara za svakog pojedinog korisnika. To se sve odvija ako OnDataChange metoda prima instancu DataSnapshot koja sadrži podatke s lokacije Firebase baze podataka. Slušatelj prima DataSnapshot koji sadrži podatke na navedenom mjestu u bazi podataka u vrijeme događaja. Pozivanje getValue() na snimku vraća prikaz podataka Java objekta. Ako na lokaciji ne postoje podaci, pozivanje getValue() vraća null. U ovom primjeru, ValueEventListener također definira metodu onCancelled() koja se poziva ako se čitanje otkaže. Na primjer, čitanje se može otkazati ako klijent nema dopuštenje za čitanje s lokacije Firebase baze podataka. Ovoj metodi se prosljeđuje objekt DatabaseError koji pokazuje zašto je došlo do kvara. U svojoj if naredbi ako instanca vec postoji klikom na igru unutar *recyclerview-a* ažurira se količina igre odnosno povećava se za jedan, povećava ukupnu cijenu igara sadržanih unutar virtualne košarice. *userCart.child(gameModel.key!!).updateChildren* obavlja ažuriranje i obaviještava o uspješnom ažuriranju. Unutar else naredbe imamo slučaj prvog biranja određene igre odnosno ako ne postoji instanca igre unutar virtualne košarice. Svi podaci iz GameModela (slika 4.9) se spremaju u CartModel (slika 4.10) te ponovno userCart poziva child(gameModel.key!!) no ovaj put setValue(cartModel) te sprema podatke u virtualnu košaricu. Slike implementacija ovih kodova su prikazane ispod.

```
private fun addToCart(gameModel: GameModel) {
        val userCart = FirebaseDatabase.getInstance().getReference( path: "Cart").child(FirebaseAuth.getInstance().currentUser!!.uid)
        userCart.child(gameModel.name!!).addListenerForSingleValueEvent(object: ValueEventListener{
            override fun onDataChange(snapshot: DataSnapshot) {
                if(snapshot.exists())
                {
                    val cartModel=snapshot.getValue(CartModel::class.java)
                    val updateData: MutableMap<String,Any> = HashMap()
                    cartModel!!.quantity=cartModel.quantity+1
                    updateData["quantity"]=cartModel.quantity
                    updateData["totalPrice"]=cartModel.quantity * cartModel.price!!.toFloat()


                    userCart.child(gameModel.name!!).updateChildren(updateData).addOnSuccessListener {  it: Void!
                        EventBus.getDefault().postSticky(UpdateCartEvent())
                        onLoadCartMessage("Uspješno ažurirana košarica")
                    }
                        .addOnFailureListener{e-> onLoadCartMessage(e.message) }
                }
                else{
                    val cartModel = CartModel()
                    cartModel.key=gameModel.key
                    cartModel.name=gameModel.name
                    cartModel.image=gameModel.image
                    cartModel.price=gameModel.price
                    cartModel.quantity=1
                    cartModel.totalPrice=gameModel.price!!.toFloat()

                    userCart.child(gameModel.name!!).setValue(cartModel).addOnSuccessListener {  it: Void!
                        EventBus.getDefault().postSticky(UpdateCartEvent())
                        onLoadCartMessage("Uspješno dodano u košaricu")
                    }
                        .addOnFailureListener{e-> onLoadCartMessage(e.message) }
                }
            }

            override fun onCancelled(error: DatabaseError) {
                onLoadCartMessage(error.message)
            }
        })
```

Programski kod 4.7 addToCart

```
package hr.ferit.deniivancok.kotlincartfirebase.model

class GameModel {
    var key:String?=null
    var name:String?=null
    var image:String?=null
    var price:String?=null
    var description:String?=null
}
```

Programski kod 4.8 klasa podataka modela igara

```
package hr.ferit.deniivancok.kotlincartfirebase.model

class CartModel {
    var key:String?=null
    var name:String?=null
    var image:String?=null
    var price:String?=null
    var quantity = 0
    var totalPrice = 0f
}
```

Programski kod 4.9 klasa podataka

4.2.5 MyCartAdapter

MyCartAdapter služi za prikaz *RecyclerView-a* unutar *CartActivity* odnosno same košarice. Bitno je naglasiti *OnBindViewHolder* metodu u kojoj se korisniku na klik tipke za brisanje otvara *AlertDialog* (slika 4.11) . Dijaloški okvir upozorenja poseban je dijaloški okvir koji se prikazuje u grafičkom korisničkom sučelju kada se dogodilo nešto što zahtijeva trenutno korisničko djelovanje.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyCartViewHolder {
    return MyCartViewHolder(LayoutInflater.from(context).inflate(R.layout.layout_cart_item,parent, attachToRoot: false))
}

override fun onBindViewHolder(holder: MyCartViewHolder, position: Int) {
    Glide.with(context).load(cartModelList[position].image).into(holder.imageView!!)
    holder.txtName!!.text= StringBuilder().append(cartModelList[position].name)
    holder.txtPrice!!.text=StringBuilder().append(cartModelList[position].price)
    holder.txtQuantity!!.text=StringBuilder().append(cartModelList[position].quantity)
    holder.btnMinus!!.setOnClickListener{ minusCartItem(holder,cartModelList[position])}
    holder.btnPlus!!.setOnClickListener{ plusCartItem(holder,cartModelList[position])}
        holder.btnDelete!!.setOnClickListener {_ ->
            val dialog= AlertDialog.Builder(context).setTitle("Obrisati stavku").setMessage("Želite li zaista obrisati igru iz košarice?")
            .setNegativeButton( text: "CANCEL"){dialog,_ -> dialog.dismiss()}
            .setPositiveButton( text: "DELETE"){dialog,_ ->
                notifyItemRemoved(position)
                FirebaseDatabase.getInstance()
                    .getReference( path: "Cart")
                    .child(FirebaseAuth.getInstance().currentUser!!.uid)
                    .child(cartModelList[position].key!!)
                    .removeValue()
                    .addOnSuccessListener { EventBus.getDefault().postSticky(UpdateCartEvent()) }
            }
            .create()
            dialog.show()
        }
}
```

Programski kod 4.10 implementacija dijaloškog okvira unutar onBindViewHolder

setTitle() postavlja naslov a *setMessage()* poruku dijaloškog okvira korisniku. Postavljanjem tipke za gašenje dijaloškog okvira metodom *setNegativeButton()* i *setPositiveButton()* za provedbu navedenih funkcija koje se nalaze u njegovom tijelu, u našem slučaju brisanje stavke video igre kao što je i prikazano na slici. Ovdje se također poziva na trenutno prijavljenog korisnika kako se ne bi obrisali podaci ostalih korisnika.

```
    private fun plusCartItem(holder: MyCartViewHolder, cartModel: CartModel) {
        cartModel.quantity += 1
        cartModel.totalPrice=cartModel.quantity * cartModel.price!!.toFloat()
        holder.txtQuantity!!.text= StringBuilder( str: "").append(cartModel.quantity)
        updateFirebase(cartModel)
    }

    fun updateFirebase(cartModel: CartModel) {
        FirebaseDatabase.getInstance().getReference( path: "Cart").child(FirebaseAuth.getInstance().currentUser!!.uid).child(cartModel.key!!)
            .setValue(cartModel)
            .addOnSuccessListener { EventBus.getDefault().postSticky(UpdateCartEvent()) }
    }
        fun minusCartItem(holder: MyCartViewHolder, cartModel: CartModel){
        if(cartModel.quantity > 1)
        {
            cartModel.quantity -=1
            cartModel.totalPrice=cartModel.quantity * cartModel.price!!.toFloat()
            holder.txtQuantity!!.text= StringBuilder( str: "").append(cartModel.quantity)
            updateFirebase(cartModel)

        }
    }
}
```

Programski kod 4.11 plusCartItem,minusCartItem i updateFirebase metode

Na slici su jos prikazane metode *minusCartItem* i *plusCartItem* koje dodaju ili oduzimaju količinu stavke video igre unutar košarice direktno unutar *CartActivity* tako što metodi *updateFirebase()* predaju instancu ažuriranog cartModel-a koja provodi ažuriranje podataka za trenutno prijavljenog korisnika prikazano na slici 4.12.

4.2.6 Učitavanje podataka iz Firebase baze podataka

Bitno je naglasiti *loadGameFromFirebase() (slika 4.13)* metodu za pristup podacima video-igara spremljenim unutar baze podataka koja se nalazi u svakoj pojedinoj aktivnosti koje predstavljaju pojedine žanrove igara.

```
private fun loadGameFromFirebase() {
    val gameModels: MutableList<GameModel> = ArrayList()
    FirebaseDatabase.getInstance().getReference( path: "Fighting/Fighting")
        .addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                if (snapshot.exists()) {
                    for (gameSnapshot in snapshot.children) {
                        val gameModel = gameSnapshot.getValue(GameModel::class.java)
                        gameModel!!.key = gameSnapshot.key
                        gameModels.add(gameModel)
                    }
                    gameLoadListener.onGameLoadSuccess(gameModels)
                } else
                    gameLoadListener.onGameLoadFailed("Igra ne postoji")
            }

            override fun onCancelled(error: DatabaseError) {
                gameLoadListener.onGameLoadFailed(error.message)
            }
        })
}
```

Programski kod 4.12 loadGameFromFirebase metoda

U ovome primjeru je prikazana *FightActivity* odnosno aktivnost koja predastavlja žanr borilačkih video igara. Kod prikazuje pristupanje spremljenim podacima video igara unutar realtime baze podataka opet uz pomoć već spomenutog *DataSnapshot-a* koji pristupa podacima iz baze podataka. Sve spremljene podatke igara sprema u listu koja biva proslijeđena metodi *onGameLoadSuccess() prikazano na slici 4.14* koja ih učitava u *recyclerview* putem adaptera .

```
    private fun init() {
        gameLoadListener = this
        kategorije.layoutManager = LinearLayoutManager( context: this@FightIgre, RecyclerView.VERTICAL,  reverseLayout: false)
    }


    override fun onGameLoadSuccess(gameModelList: List<GameModel>?) {
        val adapter = MyGameAdapter( context: this, gameModelList!!)
        kategorije.adapter = adapter
    }

    override fun onGameLoadFailed(message: String?) {
        Toast.makeText( context: this, message!!, Toast.LENGTH_LONG).show()
    }
}
```

Programski kod 4.13 učitavanje stavki igara unutar RecyclerView-a

onGameLoadSuccess() metoda predaje listu adapteru kako bi nastao odgovarajući recyclerview borilačkih video igara dok *onGameLoadFailed()* služi za prikaz poruka korisniku. Obje metode se implementiraju prilikom nasljeđivanja sučelja *IGameLoadListener (slika 4.15)* , te je tako realiziran prikaz za sve pojedine žanrove video igara.

```
import hr.ferit.deniivancok.kotlincartfirebase.model.GameModel

interface IGameLoadListener {
    fun onGameLoadSuccess(gameModelList: List<GameModel>?)
    fun onGameLoadFailed(message:String?)
}
```

Programski kod 4.14 sučelje IGameLoadListener

4.2.7. EventBus

EventBus je biblioteka otvorenog koda za Android i Javu koja koristi obrazac izdavač/pretplatnik za labavo povezivanje te omogućuje središnju komunikaciju s odvojenim klasama. *loadCartFromFirebase() prikazana na slici 4.16* je metoda unutar *CartActivity* koja predstavlja košaricu i vrlo je slične implementacije poput metode *loadGameFromFirebase().*

```
private fun loadCartFromFirebase() {
    val cartModels: MutableList<CartModel> = ArrayList()
    FirebaseDatabase.getInstance().getReference( path: "Cart")
        .child(FirebaseAuth.getInstance().currentUser!!.uid)
        .addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                for (cartSnapshot in snapshot.children) {
                    val cartModel = cartSnapshot.getValue(CartModel::class.java)
                    cartModel!!.key = cartSnapshot.key
                    cartModels.add(cartModel)
                }
                cartLoadListener!!.onLoadCartSuccess(cartModels)
            }

            override fun onCancelled(error: DatabaseError) {
                cartLoadListener!!.onLoadCartFailed(error.message)
            }

        })
}
```

Programski kod 4.15 metoda loadCartFromFirebase

```
fun onLoadCartSuccess(cartModelList: List<CartModel>) {
    var sum = 0.0
    for (cartModel in cartModelList) {
        sum += cartModel.totalPrice
    }
    if (sum == 0.0) {
        txtTotal.text = "Košarica prazna"
        kn.text = ""
    } else
        txtTotal.text = sum.toString()

    cartLoadListener!!.buyProducts(cartModelList)

    val adapter = MyCartAdapter( context: this, cartModelList)
    recycler_cart!!.adapter = adapter
}
```

Programski kod 4.16 metode onLoadCartSuccess I onLoadCartFailed

Međutim kako bi zapravo bilo moguće izmjeniti podatke košarice iz bilo koje aktivnosti to jest pritiskom *recyclerview-a* potrebno je implementirati eventbus koji će nam omogućiti komunikaciju i prijenos podataka između klasa.

```
override fun onStart() {
    super.onStart()
    EventBus.getDefault().register( subscriber: this)
}

override fun onStop() {
    super.onStop()
    if (EventBus.getDefault().hasSubscriberForEvent(UpdateCartEvent::class.java))
        EventBus.getDefault().removeStickyEvent(UpdateCartEvent::class.java)
    EventBus.getDefault().unregister( subscriber: this)
}

@Subscribe(threadMode = ThreadMode.MAIN, sticky = true)
public fun onUpdateCartEvent(event: UpdateCartEvent) {
    loadCartFromFirebase()
}
```

Programski kod 4.17 onStart, onStop i onUpdateCartEvent pretplatnička metoda

Ovo je prikaz koda koji se nalazi unutar *CartActivity klase*. Potrebno je u metodi onStart() registrirati trenutnu aktivnost pomoću linije *EventBus.getDefault().register()* te unutar metode onStop() prekinuti aktivnost kao registriranu kako bi ju po potrebi mogli zamjeniti sa drugom. No najbitniji dio se odvija u takozvanim "pretplatničkim metodama" (slika 4.18). Postavili smo *loadCartFromFirebase()* unutar pretplatničke metode kako bi svaki događaj odnosno promjena košarice bila na dohvat svake druge klase. To nam omogućava da unutar metoda svih adaptera pozivom linije koda *EventBus.getDefault().postSticky(UpdateCart)* u tijelu *addOnSuccessListenera* ,kao što je već prikazano u spomenutim *addToCart* ,*updateFirebase* i *onBindViewHolder* (unutar MyCartAdapter) metodama, ažuriramo odnosno objavimo podatke košarice iz različitih klasa neovisno u kojoj aktivnosti se korisnik trenutno nalazio.

4.2.8 Završetak kupovine-funkcija buyProducts

Ovdje se nalazi prikaz koda vrlo slične implementacije kao i one koja se nalazi unutar *onBindViewHolder-a MyCartAdaptera.* Naime implementacija još jednog *AlertDialoga* koji potvrđuje ili poništava kupnju svih proizvoda dodanih unutar košarice i time je završena kupovina te objašnjenje programskih rješenja ove aplikacije.

```
private fun buyProducts(cartModelList: List<CartModel>) {
    kupi!!.setOnClickListener { _ ->
        val dialog = AlertDialog.Builder( context: this).setTitle("Kupovina")
            .setMessage("Želite li zaista kupiti proizvode iz košarice?")
            .setNegativeButton( text: "CANCEL") { dialog, _ -> dialog.dismiss() }
            .setPositiveButton( text: "BUY") { dialog, _ ->
                FirebaseDatabase.getInstance()
                    .getReference( path: "Cart")
                    .child(FirebaseAuth.getInstance().currentUser!!.uid)
                    .removeValue()
                    .addOnSuccessListener {  it: Void!
                        EventBus.getDefault().postSticky(UpdateCartEvent())
                        Toast.makeText( context: this, text: "Uspješno obavljena kupovina!", Toast.LENGTH_SHORT).show()
                    }
            }
            .create()
        dialog.show()
    }
}
```

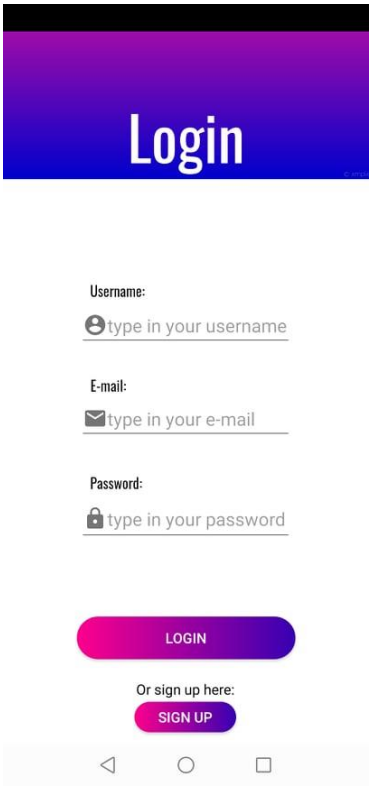
Programski kod 4.18 buyProducts metoda

5. PRIKAZ RADA I ANALIZA KORISNIČKIH SLUČAJEVA

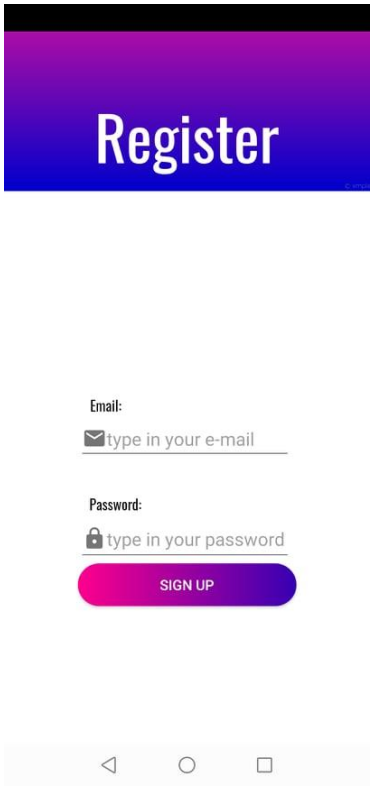
Ovim poglavljem prikazan je način rada izrađene aplikacije od registracije i prijave sve do odabira stavki i kupnje istih putem implementirane virtualne košarice. Navedena su i analizirana dva korisnička slučaja korištenja aplikacije radi provjere ispravnosti njenog rada.

5.1 Prikaz rada aplikacije

Prikazano je korištenje aplikacije i objašnjenja svih funkcionalnosti, od unosa osobnih podataka korisnika, do odabira željenog žanra video igre ,dodavanja u virtualnu košaricu, povećanje ili smanjenja količine proizvoda te uklanjanje proizvoda iz košarice. Pri pokretanju aplikacije otvara se početni zaslon prijave korisnika pokazano slikom 5.1. Ako korisnik nema korisnički profil aplikacija ga upućuje na aktivnost za registraciju prikazano slikom 5.2.



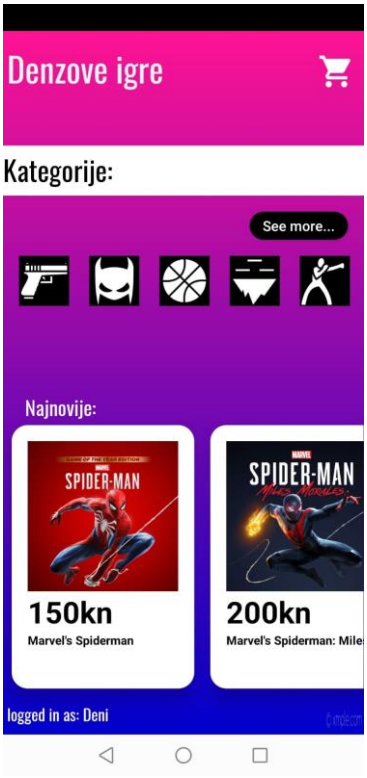
Slika 5.1 aktivnost prijave



Slika 5.2 aktivnost registracije

Nakon uspješne registracije i prijave korisnik je upućen na MainActivity gdje mu je prikazan glavni izbornik (slika 5.3) sa žanrovima igara,tipka za prikaz više žanrova, ponuda sa najnovijim igrama te tipka za odlazak na virtualnu košaricu.

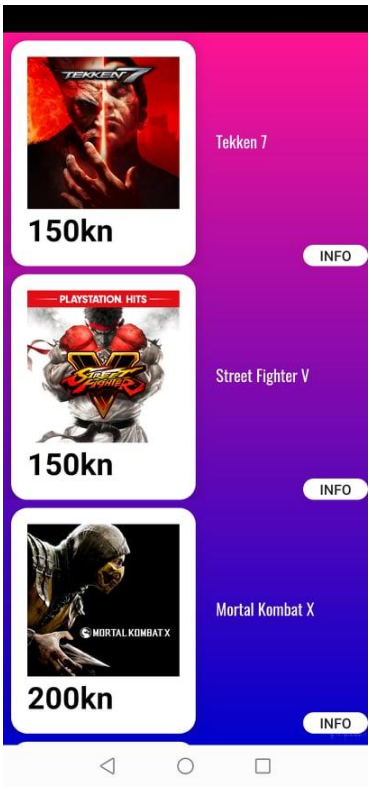
Odavdje korisnik ima odmah mogućnost odabrati jednu od najnovijih igara u ponudi ili otići na specifičnu kategoriju igara. Ako mu predstavljeno izlistanje kategorija koje se nalazi na glavnome izborniku nije dovoljno ,može pritiskom na tipku see more vidjeti sve dostupne kategorije prikazano slikom 5.4.



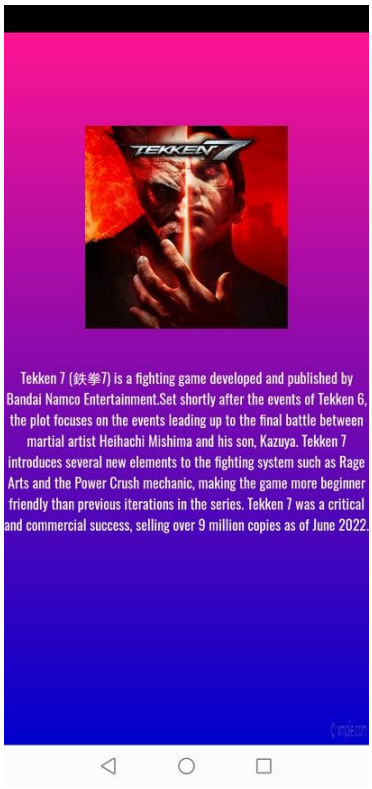
Slika 5.3 Glavni izbornik aplikacije

Slika 5.4 Izbornik kategorija aplikacije

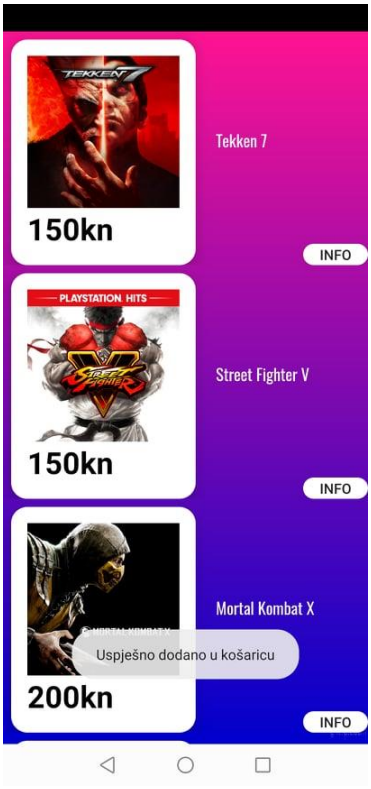
Nakon što je korisnik odabrao željenu kategoriju otvara se *recyclerview* prikaz svih igara tog žanra (slika 5.5). Korisnik klikom na info pored svake igre može dobiti dodatne informacije o samoj igri (slika 5.6) te pritiskom na sami *recyclerview* prikaz igre, dodaje ju u košaricu i o tome biva obaviješten prikazano slikom 5.7. Također ponovnim pritiskom na određenu igru više puta ažurira njenu količinu unutar košarice prikazano slikom 5.8.



Slika 5.5 Izbornik igara žanra figthting games



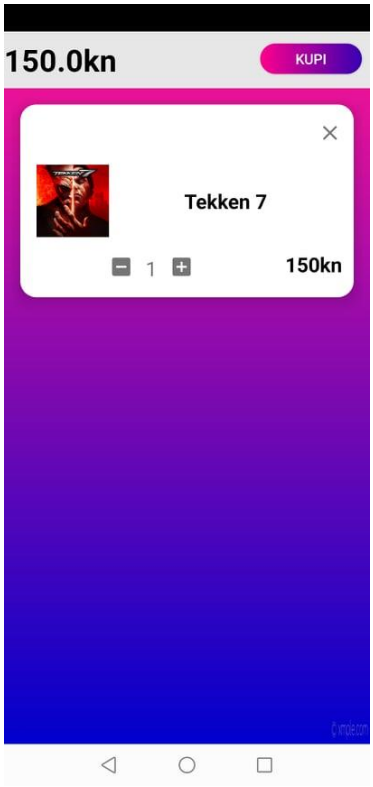
Slika 5.6 prikaz dodatnih informacija o igri



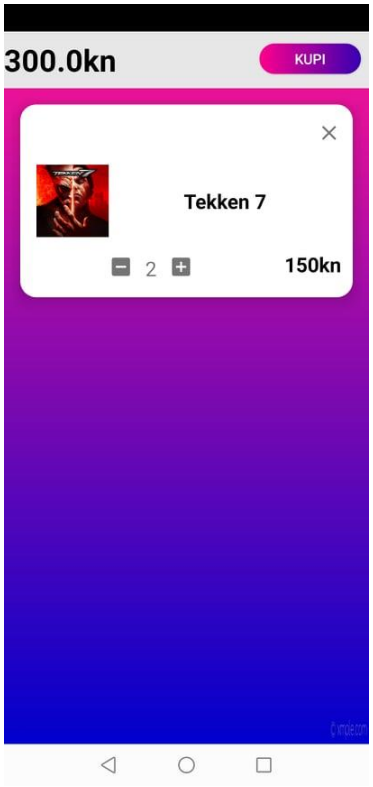
Slika 5.7 prikaz uspješnog dodavanja stavke u košaricu pritiskom na stavku

Slika 5.8 prikaz uspješnog ažuriranja stavke u košarici ponovnim pristiskom na stavku

Pritiskom na ikonu košarice na glavnom izborniku otvara nam se prikaz košarice (slika 5.9) sa svim stavkama koje smo dodali u nju i zbroj cijena svih igara trenutno u košarici. Pritiskom na tipke plus i minus ažuriramo količinu stavke te ukupnu cijenu plaćanja prikazano slikom 5.10.

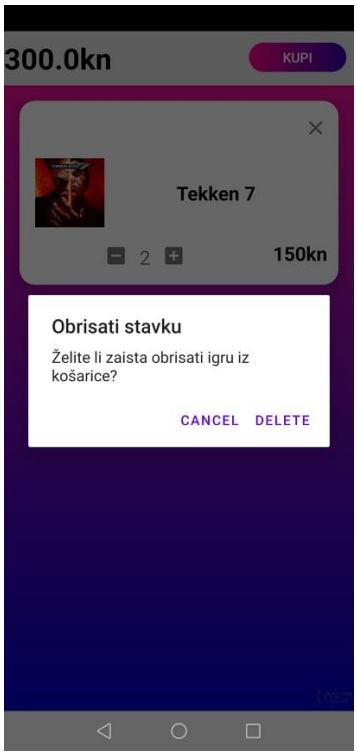


Slika 5.9 prikaz aktivnosti košarice

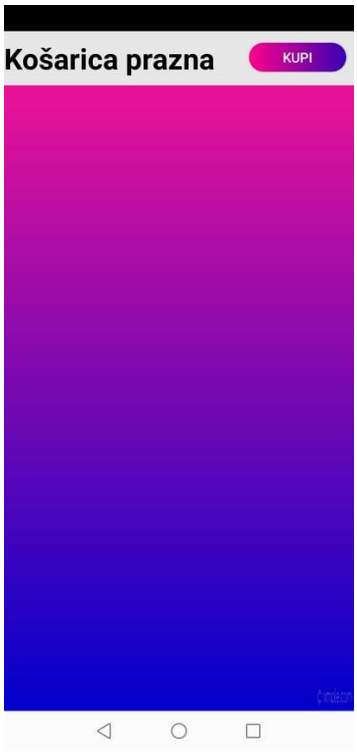


Slika 5.10 prikaz dodavanja količine stavke unutar košarice

dok na pritisak križića u gornjem desnom kutu stavke nam se otvara dijaloški okvir (slika 5.11) s kojim potvrđujemo ili otkazujemo brisanje određene stavke.

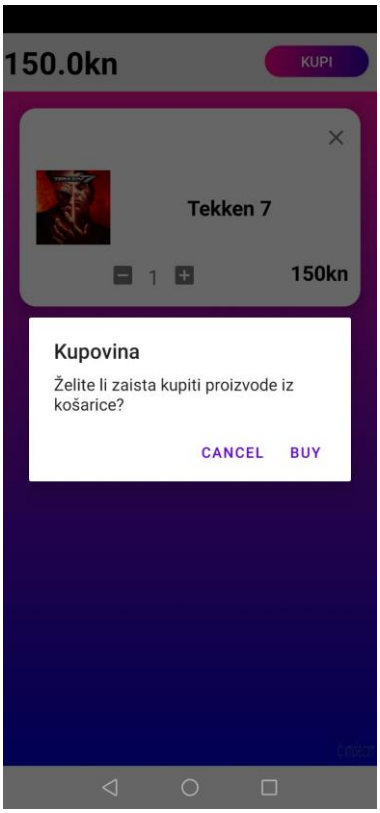


Slika 5.11 prikaz dijaloškog okvira

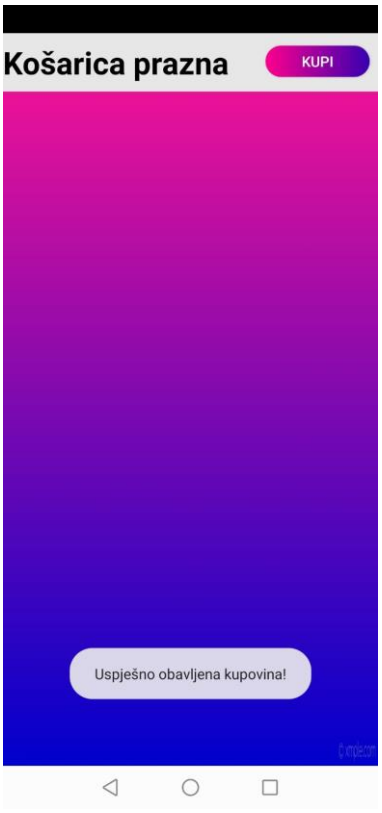


Slika 5.12 prikaz uklonjene stavke i prazne košarice

Također ako se korisnik odluči za kupnju može pristiskom na tipku kupi, otvorit će također dijaloški okvir za potvrdu ili otkazivanje kupovine svih stavki koje je dodao unutar košarice, kupiti te tako dovršiti kupovinu. Daljna implementacija dodavanja načina plaćanja je nepotrebna kako je utvrđeno po dogovoru sa mentorom.



Slika 5.13 prikaz dijaloškog okvira kupovine

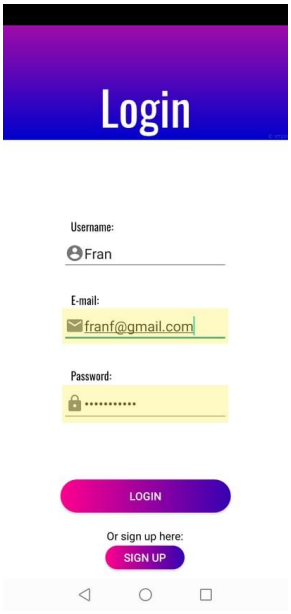


Slika 5.14 prikaz obavljene kupovine

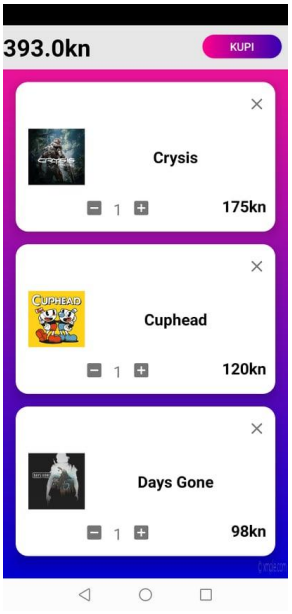
5.2 Ispitivanje rada aplikacije

5.2.1 Korisnički slučajevi

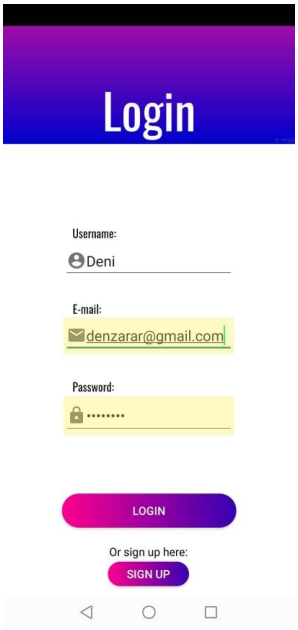
Dva različita korisnika su se prijavila u aplikaciju i dodali različite stavke unutar košarice. Slučaj prvog korisnika je prikazan slikom 5.15 i slikom 5.16 a drugog slikom 5.17 i 5.18.



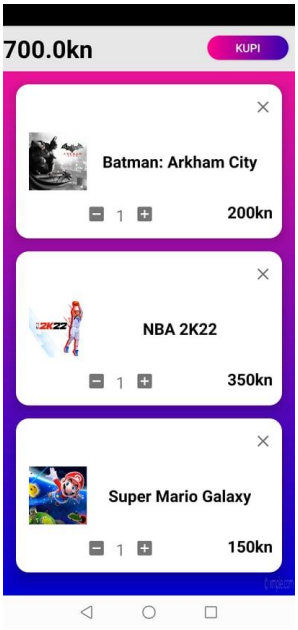
Slika 5.15 prijava-korisnički slučaj 1



Slika 5.16 virtualna košarica slučaja 1



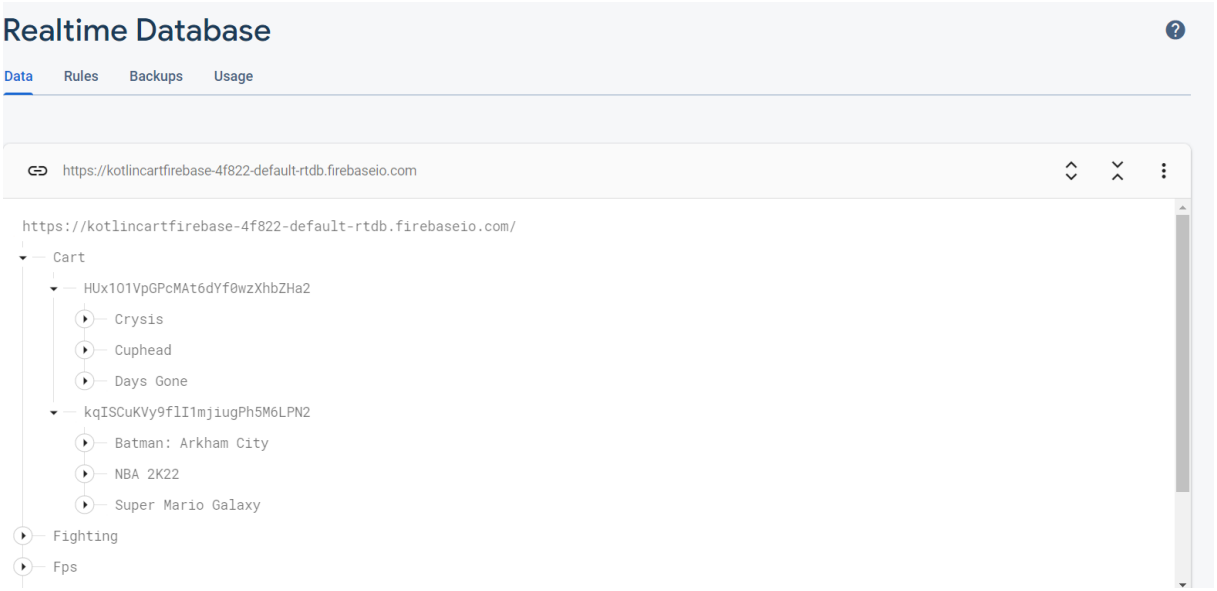
Slika 5.17 prijava-korisnički slučaj 2



Slika 5.18 virtualna košarica slučaja 2

5.2.2 Analiza korisničkih slučajeva

Ovim dvama primjerima pokazujemo da smo postigli vjerojatno i najbitniju funkciju virtualne košarice, a to je da se unutar baze podataka podaci video igara spremaju odvojeno za svakog pojedinog korisnika pod njihovu uid putanju i nemaju međusoban pristup podacima kao što možemo vidjeti na prikazu baze podataka u stvarnom vremenu na slici 5.17.



Slika 5.17 Prikaz Json realtime baze podataka

6. ZAKLJUČAK

Izrađena mobilna Android aplikacija omogućuje korisniku odabir stavki video igara te dodavanje istih unutar košarice, prikaz informacija o igrama, mijenjanje količine stavki te po želji brisanje stavki iz košarice i naravno kupovinu stavki. Aplikacija spaja sve funkcioanlnosti u vrlo jednostavno korisničko sučelje koje je lako koristiti. Aplikacija je razvijena u razvojnoj okolini Android Studio, korištenjem jezika XML i Kotlin. XML korišten je za definiranje strukture i semantike stvorenih zaslona aplikacije, a Kotlin je korišten za prilagodbu elemenata za prikazivanje korisniku te upravljanje podacima iz baze podataka i sve ostale funkcionalnosti aplikacije. Dohvaćanje i upisivanje podataka te autentifikacija korisnika je ostvarena pomoću Google-ove Firebase platforme. Nakon završenog postupka izrade mobilne aplikacije, ona je ispitana za određene slučajeve korištenja te je ispitivanje potvrdilo ispravnost rada aplikacije.

LITERATURA

[1] »Coronavirus pandemic adds $219 billion to US commerce sales in 2020-2021« Digital Commerce 360, 25 ožujak 2020.[Mrežno]. Dostupno na:https://www.digitalcommerce360.com/article/coronavirus-impact-online-retail/[posjećeno 8. lipanj 2022.].

[2] »What is Steam« PC games for Steam, 18 lipanj 2018.[Mrežno]. Dostupno na: https://pcgamesforsteam.com/what-is-steam [posjećeno 8. lipanj 2022.].

[3] »We are Gameflip« Gameflip , 19. Svibnja 2001. [Mrežno]. Dostupno na: https://gameflip.com/about/company [posjećeno 8.lipanj 2022.].

[4] »What is the PlayStationNetwork (PSN) « Lifewire, 27.Travanj 2007. [Mrežno]. Dostupno na: https://www.lifewire.com/the-playstation-network-psn-817483 [posjećeno 20.kolovoz 2022.].

[5] »Google Play« How Google Play works, 6. Ožujak 2012. [Mrežno]. Dostupno na: https://play.google.com/about/howplayworks/ [posjećeno 20. kolovoz 2022.].

[6] »What is Amazon« Amazon, 8 lipanj 2022. [Mrežno]. Dostupno na: https://www.techtarget.com/whatis/definition/Amazon [posjećeno 20. kolovoz 2022.].

[7] »GUI Architectures« Codepath, 7 prosinac 2007. [Mrežno]. Dostupno na: https://martinfowler.com/eaaDev/uiArchs.html. [posjećeno 5. Lipanj 2022.].

[8] »GUI Architectures,« MindOrks, 7. Listopad 2019. [Mrežno]. Dostupno na: https://blog.mindorks.com/mvc-mvp-mvvm-architecture-inandroid. [posjećeno 5. Lipanj 2022.].

[9] N. Chapin, »Flowchart,« u Encyclopedia of Computer Science, John Wiley and Sons Ltd. 2003, pp. 714–716.

[10] »What is Android« Android Authority, 31 prosinac 2012. [Mrežno]. Dostupno na: https://www.androidauthority.com/what-is-android-328076/. [posjećeno 5. Srpanj 2022.]

[11] »Dalvik Virtual Machine DVM« java T point, 16. veljača 2013. [Mrežno]. Dostupno na: javatpoint.com/dalvik-virtual-machine [posjećeno 7. srpanj 2022.].

[12] »Android Architecture,« Geeks for Geeks, 26. studdeni 2009. [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/android-architecture/ [posjećeno 7. srpanj 2022.].

[13] »Java Virtual Machine DVM« java T point, 16. veljača 2013. [Mrežno]. Dostupno na: https://www.javatpoint.com/jvm-java-virtual-machine [posjećeno 7. srpanj 2022.].

[14] »What is Kotlin? The Java alternative explained« Infoworld, 12. listopad 2017. [Mrežno]. Dostupno na: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html [posjećeno 7. srpanj 2022.].

[15] »XML Introduction,« Mozilla, 18 ožujak 2004. [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction. [posjećeno 16. srpanj 2022.].

[16] »Meet Android Studio,« Android Developers, 11 veljača 2002. [Mrežno]. Dostupno na: https://developer.android.com/studio/intro. [posjećeno 16. srpanj 2022.].

[17] »Firebase API Reference« Firebase, 21. srpanj 2016. [Mrežno]. Dostupno na: https://firebase.google.com/docs/reference?authuser=0&hl=en [posjećeno 16. srpanj 2022.].

SAŽETAK

U završnom radu izrađena mobilna Android aplikacija za online kupovinu video-igara pomoću implementirane virtualne košarice. Pri tom je korištena integrirana razvojna okolina Android Studio i jezici Kotlin i XML. Aplikacija omogućuje korisniku registraciju navođenjem vlastite email adrese i zaporke te naravno prijavu na isti način. Unutar Firebase-ove NoSQL baze podataka omogućuje upravljanje dodavanjem igara koje će biti ponuđene unutar trgovine te korisniku odabir i dodavanje istih unutar košarice za obavljanje kupovine.

Ključne riječi: mobilna Android aplikacija,online trgovina ,virtualna košarica, MVC.

ABSTRACT

In the bachelor’s thesis, a mobile Android application was created for online purchase of video games using the implemented virtual shopping cart. It uses the Android Studio integrated development environment and the Kotlin and XML languages. The application allows users to register by specifying their own e-mail address and password and, of course, login in the same way. Within Firebase's NoSQL database, it enables the management of adding games that will be offered within the store, and the user selects and adds them to the shopping cart.

Keywords: mobile Android application, online store, virtual shopping cart, MVC.

POPIS PROGRAMSKIH KODOVA

POPIS SLIKA

PRILOG

Kod implementacije online trgovine iz razvojne okoline Android Studio

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 31.08.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Deni Ivančok
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4209, 24.07.2018.
OIB Pristupnika:	69509005359
Mentor:	Prof. dr. sc. Krešimir Nenadić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za online trgovinu video igara
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Kratko opisati način funkcioniranja web trgovine. Izraditi i opisati postupak izrade Android aplikacije koja će se koristiti kao online trgovina video igara. Predvidjeti mogućnost registracije novih korisnika u aplikaciji kao i pohranu korisničkih podataka na udaljenom poslužitelju baze podataka. Obaviti ručno testiranje rada izrađene aplikacije i opisati postupak testirtanja. (tema rezervirana: Deni Ivančok)
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	31.08.2022.
Datum potvrde ocjene od strane Odbora:	

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije. Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 06.09.2022.

Ime i prezime studenta:	Deni Ivančok
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4209, 24.07.2018.
Turnitin podudaranje [%]:	14

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za online trgovinu video igara**

izrađen pod vodstvom mentora Prof. dr. sc. Krešimir Nenadić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta: Deni Ivančok