# Monetizacija mobilnih aplikacija i igara

**Budić, Filip**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Filip Budić**

# Monetizacija mobilnih aplikacija i igara

**ZAVRŠNI RAD**

**Varaždin, 2024.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Filip Budić**

**JMBAG: 0016155576**

**Studij: Informacijski i poslovni sustavi**

# Monetizacija mobilnih aplikacija i igara

**ZAVRŠNI RAD**

**Mentor:**

Izv. prof. dr. sc. Zlatko Stapić

**Varaždin, rujan 2024.**

*Filip Budić*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

**Sažetak**

Ovaj završni rad bavi se temom monetizacije mobilnih aplikacija i igara. Monetizacija je proces generiranja prihoda od nekog proizvoda ili stanja. Nakon definiranja monetizacije, rad se dotiče načina monetizacije mobilnih aplikacija i igara, a kod svakog načina dan je primjer implementacije. Nakon toga, rad spominje dva poslovna modela koji se tiču monetizacije i ističe prednosti i nedostatke za svaki model. Slijedi praktični dio rada koji kratko opisuje *Braintree* kao platformu nakon čega se prikazuje implementacija funkcionalnosti plaćanja.

**Ključne riječi:** monetizacija; mobilne reklame; unutar aplikacijske kupovine; freemium model; premium model; *Braintre*

# Sadržaj

# 1. Uvod

Mobilni telefoni postali su dio svakodnevice za čovjeka. Samim time, potražnja za novim, boljim i inovativnim mobilnim aplikacijama postaje sve veća. Korisnici očekuju aplikacije koje su intuitivne, brze i prilagođene njihovim potrebama. Uz to, konkurencija na tržištu je sve veća i sam razvoj aplikacija za mobilne uređaje zahtjeva praćenje trendova i stalno prilagođavanje tehnološkim inovacijama.

Monetizacija je jedan od ključnih izazova s kojim se suočavaju izdavači. Korisnici preferiraju besplatne aplikacije, a izdavači traže najbolje i najučinkovitije načine za ostvarenje prihoda. Reklame su postale sastavni dio mobilnih aplikacija i igara budući da omogućuju izdavačima da ne naplaćuju instalaciju, a ipak generiraju prihod od svakog korisnika koji se odluči instalirati i koristiti aplikaciju. Međutim, reklame mogu narušiti korisničko iskustvo. Osim reklama, prodaja/kupnja unutar aplikacijskih proizvoda je ključan način monetizacije mobilnih aplikacija, osobito monetizacije mobilnih igara. Igre mogu nuditi virtualne proizvode koji mogu, na primjer, pomoći igraču u prelasku razina igre. Mobilne aplikacije uglavnom su besplatne za preuzimanje, a izdavatelji moraju biti odlučni o tome koji model ili koji način monetizacije će aplikacija koristiti. Cilj je osigurati održivo poslovanje i kvalitetno korisničko iskustvo.

Cilj završnog rada je opisati načine monetizacije mobilnih aplikacija i igara. Rad se najviše bavi monetizacijom uz pomoć reklama i unutar aplikacijskim kupovinama. Kod njih su spomenuti alati koji omogućuju njihovu implementaciju. Također, prikazani su i najvažniji isječci programskog koda kod implementacije. Slijedi podnaslov o naplaćivanju instalacije aplikacije koji navodi prednosti i nedostatke naplaćivanja instalacije kao i besplatne instalacije. Sljedeći dio rada spominje *Freemium* i pretplatnički model te ističe ključne prednosti i nedostatke oba modela. U dijelu monetizacije *B2B* (eng. *Business-to-business*) aplikacija, spominju se fiksna cijena, cijena prema upotrebi i cijena prema broju korisnika kao način naplaćivanja *B2B* aplikacije. Zadnji dio teorijskog dijela spominje partnerski marketing i statične reklame kao načine monetizacije mobilnih aplikacija, odnosno, igara. Praktični dio rada bavi se implementacijom plaćanja unutar aplikacije uz pomoć *Braintreeja*. Također prolazi i kroz kreiranje *Braintree* korisničkog računa te prikazuje i slike zaslona aplikacije.

# 2. Metode i tehnike rada

Kao izvori za izradu rada korišteni su članci, knjige i izvori s internetskih stranica. Nadalje, za implementaciju i prikaz korištena je službena dokumentacija s *Google developer* i *Android developers* internetskih stranica. Također, korištena je dokumentacija za *Braintree*.

Deskriptivna metoda korištena je prilikom pisanja teorijskog dijela i istraživanja literature. Ona je omogućila precizan prikaz trenutnog stanja u dijelu monetizacije mobilnih aplikacija i igara. Također, prilikom istraživanja literature koristio sam metodu kompilacije kako bih još bolje mogao prikazati i objasniti načine monetizacije. Prilikom rada na praktičnom dijelu korištena je eksperimentalna metoda jer je proces implementacije zahtijevao eksperimentiranje sa bibliotekama.

Za izradu mobilne aplikacije u praktičnom dijelu rada, korišten je programski jezik *Java*. Za slanje *HTTP* zahtjeva korišten je alat *Retrofit2*. Korištene su i biblioteke *Butterknife* i *Loombok* koji su vrlo dobri za smanjenje količine ponavljajućeg koda. Na poslužiteljskoj strani koristi se programski okvir *Laravel* za *PHP* programski jezik. Također korištena je i *Braintree DropIn* biblioteka za implementaciju samog plaćanja. Korištena su integrirana razvojna okruženja *IntelliJ IDEA,* za razvoj mobilne aplikacije i *Visual Studio Code* za poslužiteljsku stranu.

Generativna umjetna inteligencija korištena je kod izrade rada. Korišten je *GitHub Copilot Chat* za objašnjenje alata kao što su *Project Lombok* i *Butterknife*. Primjer upita za objašnjenje *Butterknifea*: „What is Butterknife“. Ovakav upit omogućio mi je brzo i apstraktano saznanje o tome što je *Butterknife. ChatGPT* korišten je za preporuku literature. Primjer korištenja *ChatGPT-a* za preporuku literature: „Can you name some books about mobile app monetization“. Također, korišten je i za primjere prijevoda nekih riječi na hrvatski jezik. Primjer upita za prijevod: „Translate to Croatian: banner ads“.

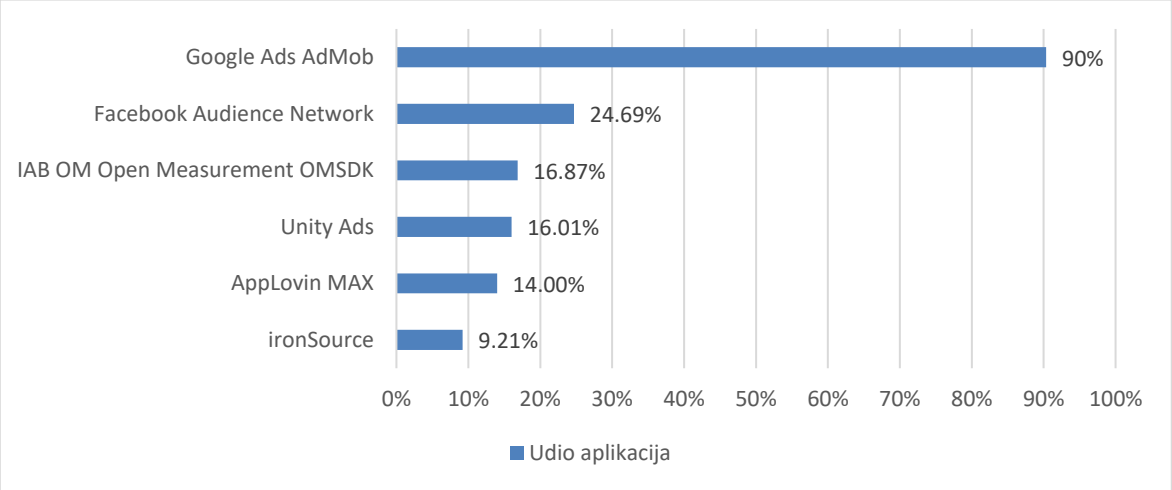
Iako je korištena generativna umjetna inteligencija prilikom izrade, sav kôd koji je sadržan u radu, napisan je ručno na temelju dokumentacije koja je dostupna o alatima i na temelju vlastitog znanja. *ChatGPT* koristio se za mogući pronalazak literature i preporuku prijevoda određenih riječi na hrvatski jezik, no nije pouzdan tako da su korišteni drugi izvori. Ovime smatram da je korištenje generativne umjetne inteligencije tokom izrade ovog rada u skladu sa smjernicama UNIZG FOI.

# 3. Monetizacija

Prema rječnicima, monetizacija je „Čin mijenjanja nečega u novac ili izražavanja nečega kao novca ili valute“ (*MONETIZATION | English meaning - Cambridge Dictionary*, 2024), dok se prema hrvatskome rječniku monetizacija definira kao „element procesa emisije novca [monetizacija duga pretvaranje javnog duga u novac]“ (*Hrvatski jezični portal*, 2024). Drugi izvori definiraju monetizaciju mobilnih aplikacija kao „Proces generiranja prihoda za podršku aplikacije“ (Content & Insights team, 2022) i „Mogućnost zarađivanja putem aplikacije“ (Salter, 2022). Prema navedenim definicijama riječi „monetizacija“, može se zaključiti da je to stvaranje zarade iz nekog proizvoda ili stanja. Prema tome, monetizacija je jako važna za mobilne aplikacije i igre, budući da izdavači žele i moraju ostvariti zaradu od aplikacija koje su objavili. Važnost generiranja zarade stoji u tome da veliki izdavači moraju platiti svoje zaposlenike, a samostalni izdavači žele dobiti nešto zauzvrat za vrijeme koje su potrošili na izradu aplikacije. Monetizacija mobilnih aplikacija i igara može se odraditi na različite načine, od naplaćivanja instaliranja mobilnih aplikacija (eng. *Cost Per Install*) do unutar aplikacijskih kupovina (eng. *In-App Purchases*).

## 3.1. Reklame unutar aplikacije

Monetizacija uz pomoć reklama ostvaruje se prikazivanjem reklama korisniku dok on koristi aplikaciju. Ovisno o alatu koji se koristi za implementaciju prikaza reklama, postoje različite vrste reklama koje se mogu prikazati korisniku. Isto kao i tipova reklama, postoji velika količina alata za implementaciju prikaza reklama unutar mobilnih aplikacija i igara. Na slici broj 1, može se vidjeti da je Google Ads AdMob najkorišteniji alat za implementaciju reklama.




Slika 1: Najpopularniji alati za implementaciju reklama u aplikacije (Ceci, 2024.)

Drugi najpopularniji alat je *Facebook Audience Network* (sada znan kao *Meta Audience Network*), a nakon njega slijede *OMSDK* i *Unity Ads*. *Unity Ads* koristi se kod mobilnih igara radi vrlo jednostavnog postavljanja i implementacije. *Meta Audience Network* svojim korisnicima pruža odabir između četiri vrste reklama, a to su: *Native ads, Banner ads, Interstitial ads* i *Rewarded Video ads* (Meta Business Help Center, 2024a). *Google Ads AdMob* pruža dvije vrste reklama više nego *Meta Audience Network*, a to su *App Open Ads* i *Rewarded Interstitial Ads* (Google AdMob, 2024a). Oba alata rade na temelju *CPM-a* (eng. *Cost Per Mile*). *CPM* se odnosi na iznos koji oglašivač plaća za 1000 prikaza oglasa (Akash Yadav, 2022); (AdRoll, 2024). *Meta Audience Network* i *Google AdMob* korisnicima isplaćuju zaradu oko 21. dana u mjesecu pri čemu minimalni iznos mora biti veći od 100 američkih dolara (Meta Business Help Center, 2024b); (Google AdMob, 2024d). Podjela prihoda kod *Google AdMoba* iznosi 60:40 gdje 40% zarade zadržava *Google*, a *Meta Audience Network* još nije izrazio fiksnu podjelu prihoda (Akash Yadav, 2022).

### 3.1.1. *Google AdMob*

Za detaljnije objašnjenje implementacije i prikaza nekoliko vrsta reklama, koristit će se *Google AdMob*. Prikazani su važni isječci koda za implementaciju reklama unutar mobilne aplikacije, a koristit će programski jezik *Kotlin*. Prije pisanja koda, potrebno je dodati ovisnost u *build.gradle* datoteku na razini aplikacije (kôd 1).

```
implementation ("com.google.android.gms:play-services-ads:$version")
```

Kôd 1: Ovisnost za *Google AdMob* (Google AdMob, 2024c)

Nakon dodavanja, potrebno je učitati promjene u *gradelu* kako bi se mogli koristiti dodaci koje smo dobili iz ovisnosti. Osim ovisnosti, potrebno je dodati i identifikator aplikacije u *AndroidManifest.xml* datoteku (kôd 2).

```
<manifest>
  <application>
    <meta-data
        android:name="com.google.android.gms.ads.APPLICATION_ID"
        android:value="ca-app-pub-xxxxxxxxxxxxxxxx~yyyyyyyyyy"/>
</application>
</manifest>
```

Kôd 2: *Manifest meta-data* (Google AdMob, 2024c)

Kako bi se dobio generirani identifikator, potrebno je prijaviti aplikaciju na *AdMob* internet aplikaciji, a potom zamijeniti desnu stranu od *android:value* s identifikatorom dodane aplikacije. U ulaznoj klasi aplikacije potrebno je inicijalizirati *Google Mobile Ads* (kôd 3). Nakon inicijalizacije odlučuje se o vrsti reklame koja se želi prikazati, te je potrebno poslati zahtjev za nju.

```
MobileAds.initialize(this, initializationStatus -> {});
```

Kôd 3: Inicijalizacija (Google AdMob, 2024c)

#### 3.1.1.1. Reklame u obliku zastave (eng. *Banner ads*)

Reklame u obliku zastave (eng. *banner ads*) su pravokutne reklame koje ostaju na ekranu dok korisnici koriste aplikaciju, te su najpopularnija i najmanje nametljiva vrsta prikaza reklama (adymob, 2024). Mogu biti postavljene na vrhu ili dnu ekrana te između sadržaja. *Google* nudi adaptivne (eng. *Adaptive*), standardne (eng. Standard) i pametne (eng. Smart) reklame u obliku zastave. Standardne mogu prikazivati samo one reklame koje su fiksne veličine. Pametne reklame više nisu podržane, pri čemu se kao zamjena koriste adaptivne reklame koje se dijele na fiksirane reklame (eng. *Anchored*) i reklame unutar sadržaja (eng. *Inline*). Kao što sam naziv govori, fiksne reklame u obliku zastave napravljene su kako bi se prikazivale samo na vrhu ili dnu ekrana, a prikazuju se tijekom upotrebe aplikacije ili dok nije završena aktivnost. S druge strane, reklame unutar sadržaja zamišljene su da se prikazuju unutar pomičnog sadržaja (eng. *scrollable content*) (Google AdMob, 2024).

Za testiranje prikaza reklama može se koristiti testni identifikacijski kod. Ako je potrebno testirati s reklamama koje se prikazuju stvarnim korisnicima, važno je dodati testni mobilni telefon i generirati identifikacijski kod za reklame u obliku zastave. U isječku koda koristit će se identifikacijski kod za testne reklame. Element koji je potrebno dodati je:

```
AdView(context).apply {
    adUnitId = ca-app-pub-3940256099942544/9214589741
    setAdSize(AdSize.BANNER)
    loadAd(AdRequest.Builder().build())
}
```

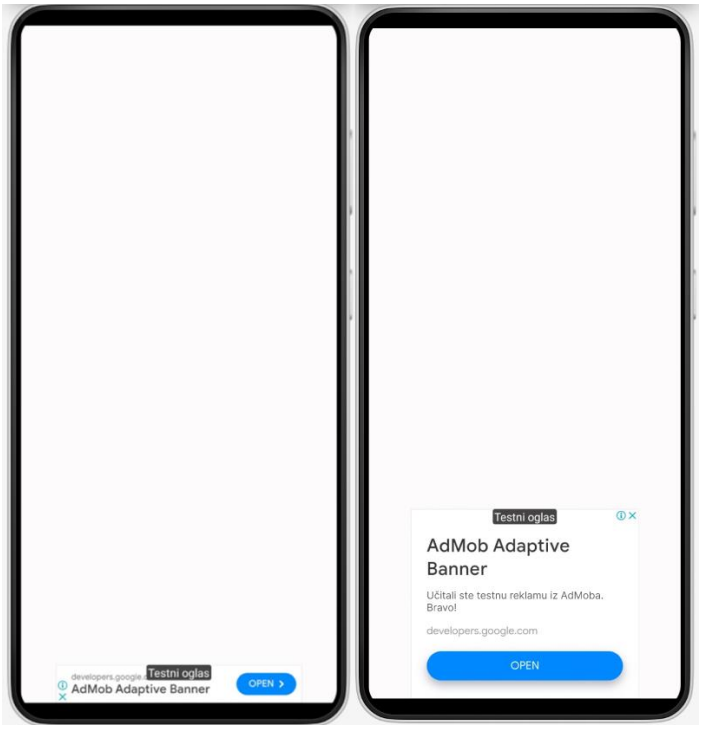
Kôd 4: Slanje zahtjeva za reklamu (Google AdMob, 2024b)

Važno je napomenuti da *Google Ads* nema podršku za *Jetpack Compose* te je reklame potrebno „omotati“ u *AndroidView* komponentu koja omogućuje prikaz *view* komponenata u *Jetpack Composeu* (kôd 5).

```
AndroidView(modifier = modifier.fillMaxWidth(), factory = { context ->
            AdView(context).apply{
                adUnitId = ca-app-pub-3940256099942544/9214589741
                setAdSize(AdSize.LARGE_BANNER)
                loadAd(AdRequest.Builder().build()
)}})
```

Kôd 5: Zahtjev za reklamu umotan u *AndroidView* (Google AdMob, 2024b)

Može se izabrati između različitih veličina i prikaza, kao što su na primjer: *LARGE_BANNER, BANNER* i *MEDIUM_RECTANGLE*. Na slici 2 prikazan je *LARGE_BANNER* (lijevo) i *MEDIUM_RECTANGLE* (desno).




Slika 2: *Adaptive Banner*

Kao što je već napomenuto, korišten je identifikacijski kod za testne reklame, pri čemu se ne prikazuju reklame koje se zapravo koriste, već se prikazuje testna reklama, što je potvrđeno natpisom „Testni oglas“ koji se nalazi na reklami (slika 2).

#### 3.1.1.2. *App open* i *Interstitial* reklame

*App open* su reklame koje se prikazuju kada korisnik otvori aplikaciju koja je već pokrenuta, ali se nalazi u memoriji. Drugim riječima, reklama se prikaže kada korisnik dovede aplikaciju u prvi plan. Takve reklame mogu se koristiti i kod otvaranja aktivnosti, no *Google AdMob* ima posebnu vrstu reklama koje se prikazuju prilikom prirodnog mijenjanja aktivnosti, a nazivaju ih *Interstitial ads*. U primjeru će biti korištena *App open* reklama, koja će se prikazivati kod otvaranja aktivnosti. Jedan od primjera interstitial reklame je prikazivanje oglasa korisniku nakon što završi razinu igre. Zapravo, ove reklame prikazuju se svaki puta, osim prvog, kada korisnik otvori aktivnost na kojoj je pripremljena reklama. *App* open reklamu moguće je prikazati i kod hladnog pokretanja (eng. *Cold start*). Preporučuje se kreirati ekran za učitavanje koji se prikazuje dok se reklama učitava kako bi se izbjeglo loše korisničko iskustvo (Google AdMob, 2024). Ove vrste reklama prikazuju se u punoj veličini zaslona. Takve vrste reklama su izdavačima jako dobre u pogledu zarade, budući da je veća zarada po prikazu. S druge strane, ove vrste reklama pogoršavaju korisničko iskustvo i često izazivaju frustraciju kod korisnika jer su jako nametljive (adymob, 2024).

Kod samog postavljanja reklama, u ovisnosti načina na koji će se koristiti reklama, potrebno je naslijediti klasu *Application* i implementirati sučelja *ActivityLifecycleCallbacks* i *LifecycleObserver*. U primjeru nisu implementirana sučelja jer je zamišljeno da se reklama prikazuje svakim otvaranjem aktivnosti.

```
class MyApplication : Application() {
    var appOpenAdManager: AppOpenAdManager? = null
    override fun onCreate() {
        super.onCreate()
        MobileAds.initialize(
            this
        ) { }
        appOpenAdManager = AppOpenAdManager(this)
    }
}
```

Kôd 6: Inicijalizacija klase *AppOpenAdMAnager* (Google AdMob, 2024)

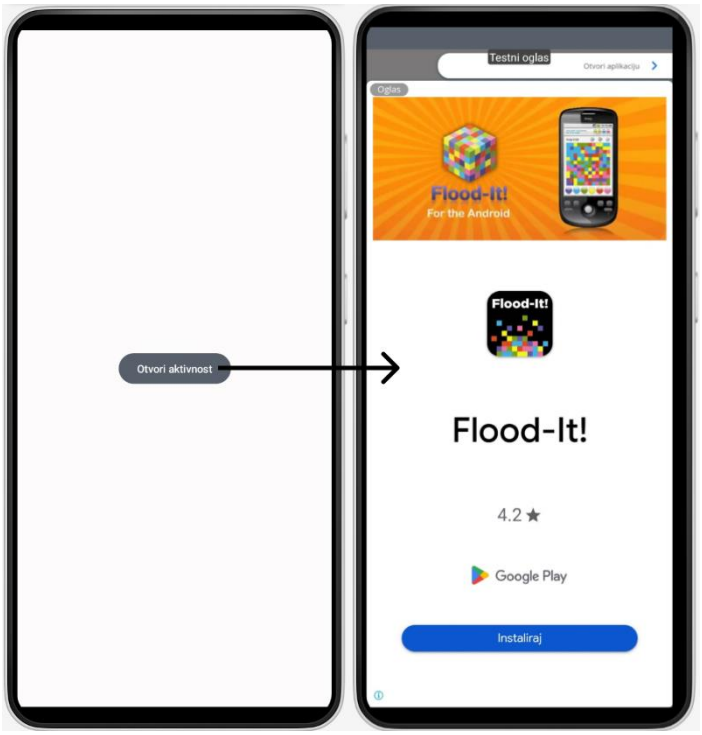
Nasljeđivanje klase *Application* omogućuje održavanje globalnog stanja aplikacije i inicijalizira se prije bilo koje druge klase kod stvaranja procesa aktivnosti. Osim toga, inicijalizira se klasa *AppOpenAdManager* koja se koristi za slanje zahtjeva za reklamu (Google AdMob, 2024). Kôd 7 prikazuje klasu *AppOpenAdManager*.

```
class AppOpenAdManager (private val context: Context) {
    private var appOpenAd: AppOpenAd? = null
    private var isLoadingAd = false
    var isShowingAd = false
    private fun loadAd() {
        if (isLoadingAd || isAdAvailable) return
        isLoadingAd = true
        val request = AdRequest.Builder().build()
        AppOpenAd.load(
            context, AD_UNIT_ID, request,
            AppOpenAd.APP_OPEN_AD_ORIENTATION_PORTRAIT,
            object : AppOpenAd.AppOpenAdLoadCallback() {
                override fun onAdLoaded(ad: AppOpenAd) {
                    appOpenAd = ad
                    isLoadingAd = false
                }

                override fun onAdFailedToLoad(loadAdError: LoadAdError) {
                    isLoadingAd = false
                }
            })
    }

    fun showAdIfAvailable(activity: Activity) {
        if (isShowingAd) return

        if (!isAdAvailable) {
            loadAd()
            return
        }
        appOpenAd!!.fullScreenContentCallback = object :
FullScreenContentCallback() {
            override fun onAdDismissedFullScreenContent() {
                appOpenAd = null
                isShowingAd = false
                loadAd()
            }

            override fun onAdFailedToShowFullScreenContent(adError: AdError) {
                Log.d(LOG_TAG, adError.message)
                appOpenAd = null
                isShowingAd = false
                loadAd()
            }
```

```
            override fun onAdShowedFullScreenContent() {}
        }
        isShowingAd = true
        appOpenAd!!.show(activity)
    }
    private val isAdAvailable: Boolean
       get() = appOpenAd != null}
```

Kôd 7: Klasa *AppOpenAdManager* (Google AdMob, 2024)

Metoda *loadAd()* učita reklamu ako ona još nije učitana. Drugim riječima, metoda *loadAd()* šalje zahtjev za reklamu te sprema reklamu u varijablu *appOpenAd*, dok metoda *showAdIfAvailable()* prikazuje reklamu ako je ona dostupna i ako se već ne prikazuje. Izgled reklame u aplikaciji vidljiv je na slici broj 3.

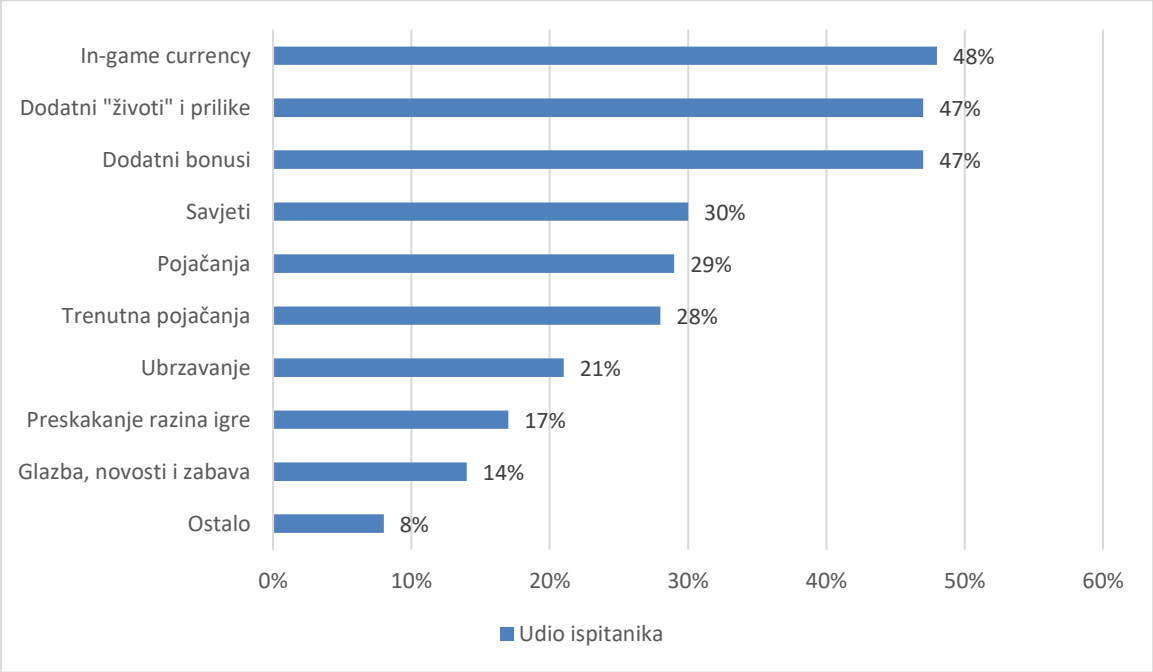



Slika 3: *App open ad*

Kao što je prikazano na slici, reklama popuni cijeli zaslon i moguće ju je zatvoriti u gornjem desnom kutu.

### 3.1.1.3. Nagradne reklame (eng. *Rewarded ads*)

Reklame se mogu koristiti i kao opcije za nagrađivanje igrača. *Google AdMob* takve reklame naziva nagradnim reklamama (eng. *Rewarded ads*). Primjer korištenja takve vrste su reklame u igri koje nagrađuju igrače s valutom koja se koristi u igri. Na slici broj 4 prikazan je graf iz kojeg se mogu vidjeti najpoželjnije nagrade igrača mobilnih igara.




Slika 4: Najpoželjnije nagrade od igrača mobilnih igara u SAD-u (Clement, 2021)

Može se uočiti da su igrači mobilnih igara iz SAD-a najviše voljni pogledati reklamu za unutar aplikacijske valute i dodatne „živote“ ili bonuse.

Nagradne reklame predstavljaju korist izdavačima, ali i igračima jer igrači imaju mogućnost odabira kada će gledati reklamu, a samim time, ova vrsta reklama se ne smatra nametljivom. Igrači na kraju gledanja dobiju nagradu koja će im pomoći unutar igre pa će se često odlučiti na gledanje reklame. U korist izdavačima ide i činjenica da se pregled nagradnih reklama plaća više nego ostatak reklama, izuzev *Interstitial* i *App open* reklama (Medina, 2024). Implementacija prikaza reklame nije previše složena, te je kao takva, dobar način monetizacije igara. Implementacija slijedi:

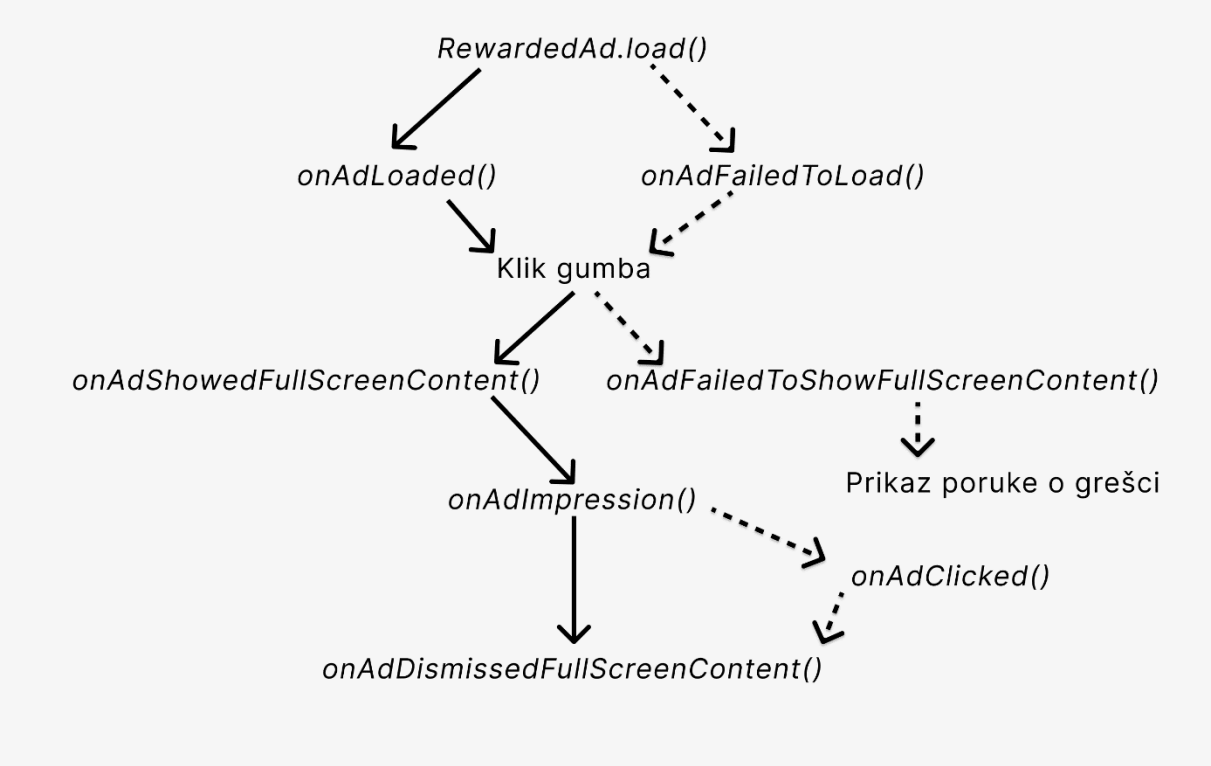
```
class RewardedAdManager(private val context: Context) {
    private var rewardedAd: RewardedAd? = null
    private val TAG = "RewardedAdManager"
```

```
    init {
        MobileAds.initialize(context) {}
        loadAd()}
    fun loadAd() {
        val adRequest = AdRequest.Builder().build()
        RewardedAd.load(context, "ca-app-pub-3940256099942544/5224354917",
adRequest, object : RewardedAdLoadCallback() {
            override fun onAdFailedToLoad(adError: LoadAdError) {
                Log.d(TAG, adError.toString())
                rewardedAd = null}
            override fun onAdLoaded(ad: RewardedAd) {
                Log.d(TAG, "Ad was loaded.")
                rewardedAd = ad
                setFullScreenContentCallback()}})}
    private fun setFullScreenContentCallback() {
        rewardedAd?.fullScreenContentCallback = object :
FullScreenContentCallback() {
            override fun onAdClicked() {
                Log.d(TAG, "Ad was clicked")}
            override fun onAdDismissedFullScreenContent() {
                Log.d(TAG, "Ad dismissed fullscreen content.")
                rewardedAd = null
                loadAd()}
            override fun onAdFailedToShowFullScreenContent(adError: AdError) {
                Log.e(TAG, "Ad failed to show fullscreen content.")
                rewardedAd = null}
            override fun onAdShowedFullScreenContent() {
                Log.d(TAG, "Ad showed fullscreen content.")}
            override fun onAdImpression() {
                Log.d(TAG, "Ad recorded an impression.")}}}
    fun showAd(onRewardEarned: (Int) -> Unit) {
        rewardedAd?.let { ad ->
            ad.show(context as MainActivity, OnUserEarnedRewardListener {
rewardItem ->
                val rewardAmount = rewardItem.amount
                val rewardType = rewardItem.type
                Log.d(TAG, "User earned the reward: $rewardAmount $rewardType")
                onRewardEarned!!(rewardAmount)
            })
        } ?: run {
            Log.d(TAG, "The rewarded ad wasn't ready yet.")
        }
    }
}
```

Kôd 8: Klasa *RewardedAdManager* (Google AdMob, 2024)

Prije objašnjenja klase *RewardedAdManager*, potrebno je znati da je na *MainActivity* postavljen gumb koji aktivira prikaz reklame. Također, postavljen je tekst koji prikazuje količinu novca koji igrač trenutno posjeduje.

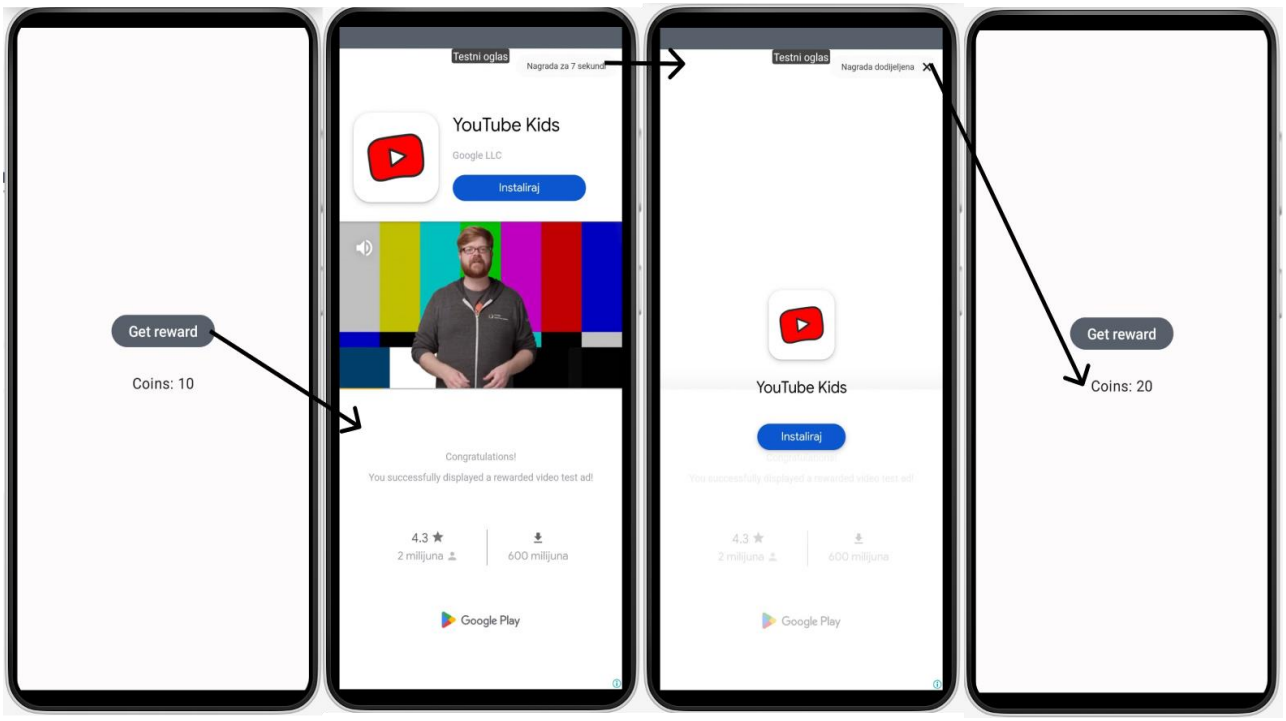
Klasa *RewardedAdManager* inicijalizira, zahtjeva i učita reklamu koja će se prikazati nakon pritiska gumba. Sam prikaz i nagrađivanje korisnika obavlja se u metodi *showAd()* koja prima lambda funkciju, naziva *onRewardEarned()*. Ona se izvršava kada igrač (korisnik) odgleda reklamu do kraja. Povratni poziv naziva *onUserEarnedRewardListener* bilježi nagradu koju je igrač zaradio i prosljeđuje iznos nagrade (Google Developers, 2024). Kod učitavanja reklame, odnosno, slanja zahtjeva za reklamu, koristi se *RewardedAdLoadCallback* koji ima dva moguća ishoda. Prvi i poželjni ishod je da se reklama spremi u varijablu *rewardedAd*, a ukoliko dođe do neke pogreške, izvršit će se funkcija *onAdFailedToLoad()* (Google Developers, 2024). *FullScreenContentCallback* omogućuje praćenje životnog vijeka reklame i praćenje interakcije korisnika s reklamom (Google Developers, 2024).




Slika 5: Životne funkcije reklame

Nakon pokretanja aktivnosti, potrebno je učitati reklamu. Ako je došlo do pogreške, izvršit će se metoda *onAdFAiledToLoad()* i ako korisnik klikne na gumb za prikaz reklame, izvršit će se metoda *onAdFailedToShowFullScreenContent()*. S druge strane, ako se reklama učita (*onAdLoaded()*) i prikaže nakon klika gumba, izvršit će se metoda

*onAdShowedFullScreenContent()*. Metoda *onAdImpression()* izvršava se kada je reklama učitana i kada je utvrđeno da je prikazana korisniku. Prilikom klika na poveznicu, koja se nalazi na reklami, izvršava se metoda *onAdClicked()*. Zadnja metoda, *onAdDismissedFullScreenContent()* se izvršava kada korisnik zatvori reklamu (Google Developers, 2024b). Slika 6 prikazuje put korisnika kroz aplikaciju.




Slika 6: *Wireframe* nagradne reklame

Na slici je vidljivo da se nakon klika na gumb otvara reklama koja prikazuje video. Ako korisnik prekine reklamu prije nego što video završi, neće dobiti nagradu. U slučaju da pogleda video do kraja, pojavljuje se mogućnost zatvaranja reklame te potom igrač dobiva nagradu. Nagrada se može uočiti u donjem lijevom kutu, pri čemu je vidljivo da igrač ima više „novaca“ nego prije gledanja reklame.

## 3.2. Unutar aplikacijske kupovine

Unutar aplikacijske kupovine (eng. *In-app purchases*) predstavlja kupnju proizvoda unutar aplikacije. Ovisno o aplikaciji, mogu se pojaviti u različitim oblicima, od kupnje značajki i funkcionalnosti do boljeg korisničkog iskustva. Godine 2016., korisnici *Android* operacijskih sustava prosječno su potrošili 0.43 američkih dolara, što je vidljivo na slici 7. Prosjek kod aplikacija za mobilne uređaje koji koriste *iOS* kao operacijski sustav bio više nego duplo veći.

Slika broj 7 prikazuje prosječnu potrošnju korisnika putem unutar aplikacijskih kupovina po aplikaciji.


Slika 7: Prosječna potrošnja korisnika na unutar aplikacijske kupovine (AppsFlyer, bez dat.)

Prema statistici iz 2024. godine, prosječna potrošnja korisnika putem unutar aplikacijskih kupovina je 8.37 američkih dolara, a prosječna potrošnja kod korisnika *Apple* mobilnih uređaja je 2.5 puta veća nego na mobilnim uređajima s *Android* operacijskim sustavima, što nastavlja trend koji je prikazan na slici 6 (Lindner, 2024). Također, važan pokazatelj da su unutar aplikacijske kupovine dobar način za monetizaciju mobilnih aplikacija i igara je taj da unutar aplikacijske kupovine generiraju dvadeset puta više zarade nego što zarađuju aplikacije koje treba platiti prije instalacije. Drugi dobar pokazatelj jest taj da kupnje unutar aplikacije čine 48,2% prihoda od mobilnih aplikacija (Lindner, 2024).

### 3.2.1. Tipovi unutar aplikacijskih kupovina

Kako navode Sharma (2023) i Apple Developer (2024), postoje četiri vrste proizvoda koje korisnici mogu kupiti unutar aplikacije, a oni su navedeni u nastavku.

Kupnja proizvoda koji se ne mogu potrošiti (eng. *Non-consumable in-app purchase*) odnosi se na proizvode koje korisnici nakon kupnje mogu nastaviti koristiti. Na primjer, aplikacija ima ponudu različitih vizualnih tema koje korisnik može kupiti i tada koristiti. Kada korisnik jednom kupi temu, on ju može koristiti uvijek. Drugim riječima, može se koristiti do kraja njegovog korištenja aplikacije bez potrebe za ponovnom kupnjom. Drugi primjer može biti kada korisnik kupi novu razinu igre koja nije dostupna u besplatnoj verziji igre. Tada korisnik zadržava tu razinu i može ju ponovo odigrati beskonačno mnogo puta. Takvi proizvodi uglavnom imaju nešto višu cijenu, budući da korisnici zauvijek zadržavaju proizvod (Sharma, 2023); (Apple Developer, 2024).

Kupnja potrošivih proizvoda (eng. *Consumable in-app purchase*) odnosi se na proizvode koji su u kontrastu s proizvodima koji se ne mogu potrošiti. Potrošive proizovde korisnik može iskoristiti određeni broj puta, a nakon toga potrebno je ponovo kupiti proizvod kako bi ga mogao dalje koristiti. Na primjer, korisnik želi kupiti proizvod koji mu omogućuje

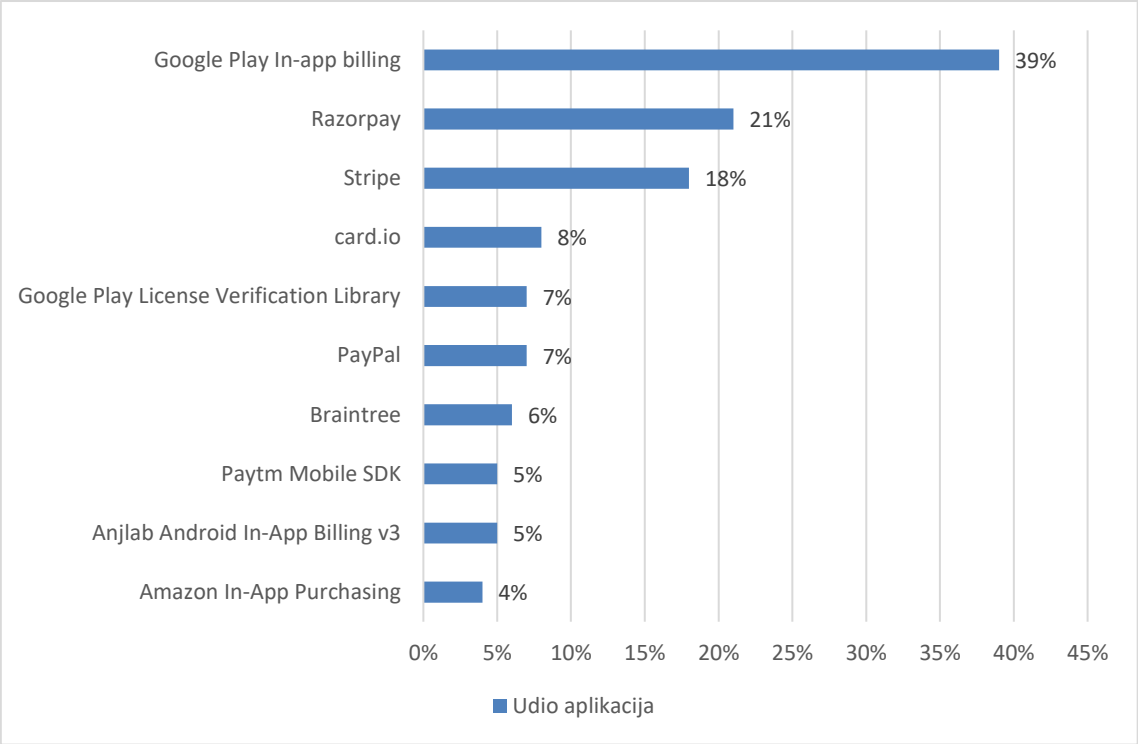
pojednostavljenje razine u igri. Nakon kupnje, korisnik može koristiti taj proizvod, no on ima ograničen broj korištenja. Nakon što korisnik iskoristi taj proizvod određen broj puta, korisnik ga neće moći ponovo koristiti. Ukoliko želi ponovo koristiti taj proizvod, potrebno je ponovno kupiti taj proizvod (Sharma, 2023); (Apple Developer, 2024).

Automatski obnovljive pretplate (eng. *Auto-Renewable Subscription*) su još jedan tip unutar aplikacijskih kupovina. One se automatski naplaćuju i mogu se naplaćivati na godišnjoj, mjesečnoj, tjednoj ili dnevnoj bazi. Kao primjer možemo uzeti aplikaciju koja korisnicima nudi značajku spremanja, ubrzavanja ili preskakanja pjesama. Kada se korisnik odluči na pretplatu, koja mu omogućuje preskakanje pjesama i slušanje pjesama bez oglasa, korisnik daje privolu da se ta transakcija ponovo izvrši na mjesečnoj bazi kako bi zadržao značajke (Sharma, 2023); (Apple Developer, 2024)

Neobnovljive pretplate (eng. *Non-Renewing Subscription*) su pretplate koje korisnik plati da bi na određeno razdoblje imao nove značajke ili poboljšano korisničko iskustvo. Nakon isteka perioda, korisnik mora ponovo ručno platiti kako bi mogao koristiti dodatne značajke (Sharma, 2023); (Apple Developer, 2024).

### 3.2.2. Alati za implementaciju unutar aplikacijskih kupovina

Postoji velik broj alata za implementaciju unutar aplikacijskih kupovina, a najpopularniji je *Google Play Billing Library* koji se koristi uz *Google Play Console*. *Google Play Console* je platforma koja omogućuje izdavačima da objave i prate performanse njihove aplikacije na *Google Play Storeu*. Na slici 8 prikazano je 10 najpopularnijih alata za implementaciju unutar aplikacijskih kupovina.

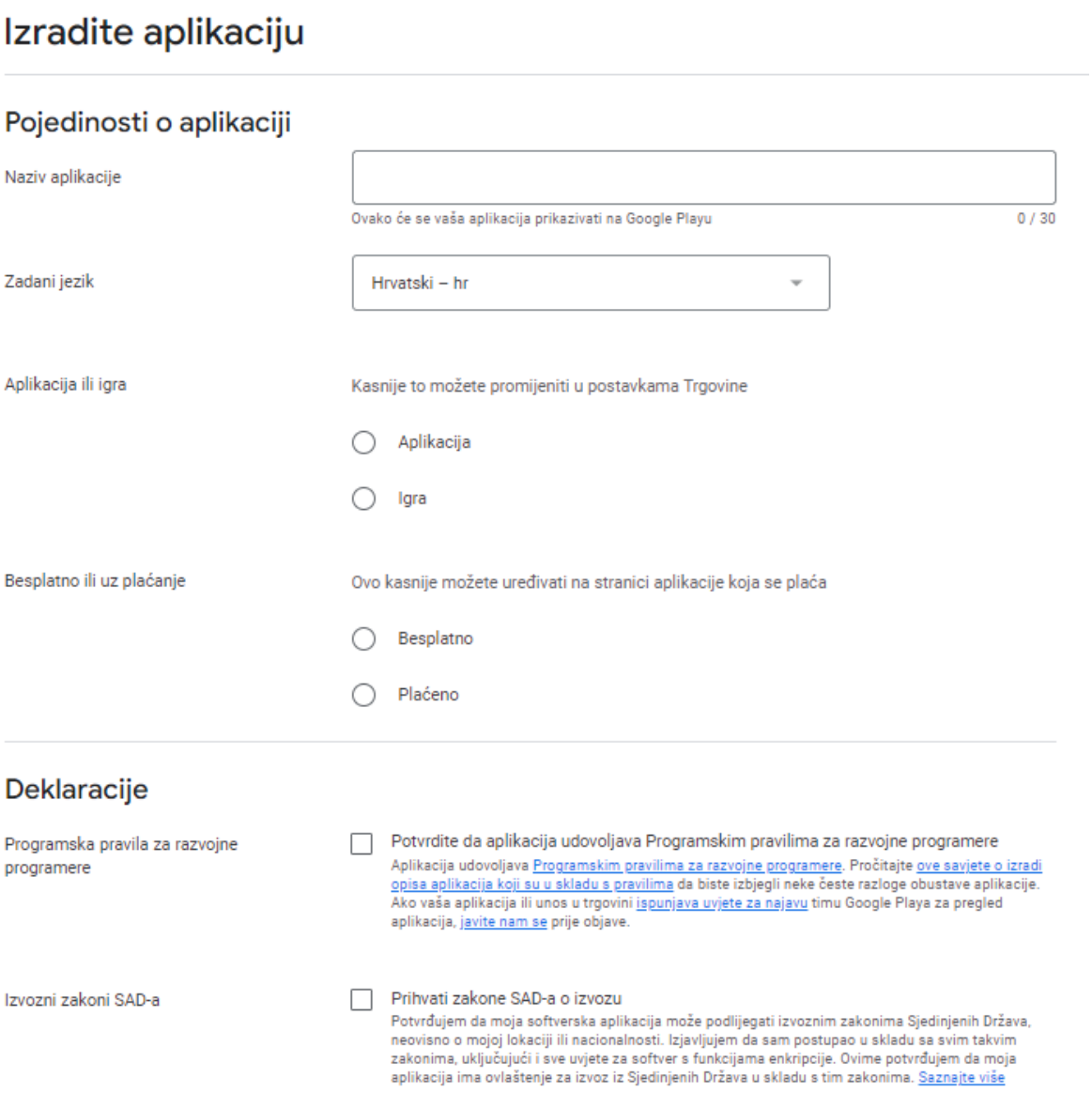



Slika 8: Popularni alati za implementaciju *IAP-a* (Ceci, 2023)

Najpopularniji alat za implementaciju unutar aplikacijskih kupovina kod aplikacija za *Android* operacijske sustave je *Google Play Billing*. Nakon njega slijede *Razorpay* i *Stripe* (42matters, 2023). U praktičnom dijelu biti će prikazana implementacija *Braintreeja*.

### 3.2.3. *Google Play Console* i *Google Play Billing*

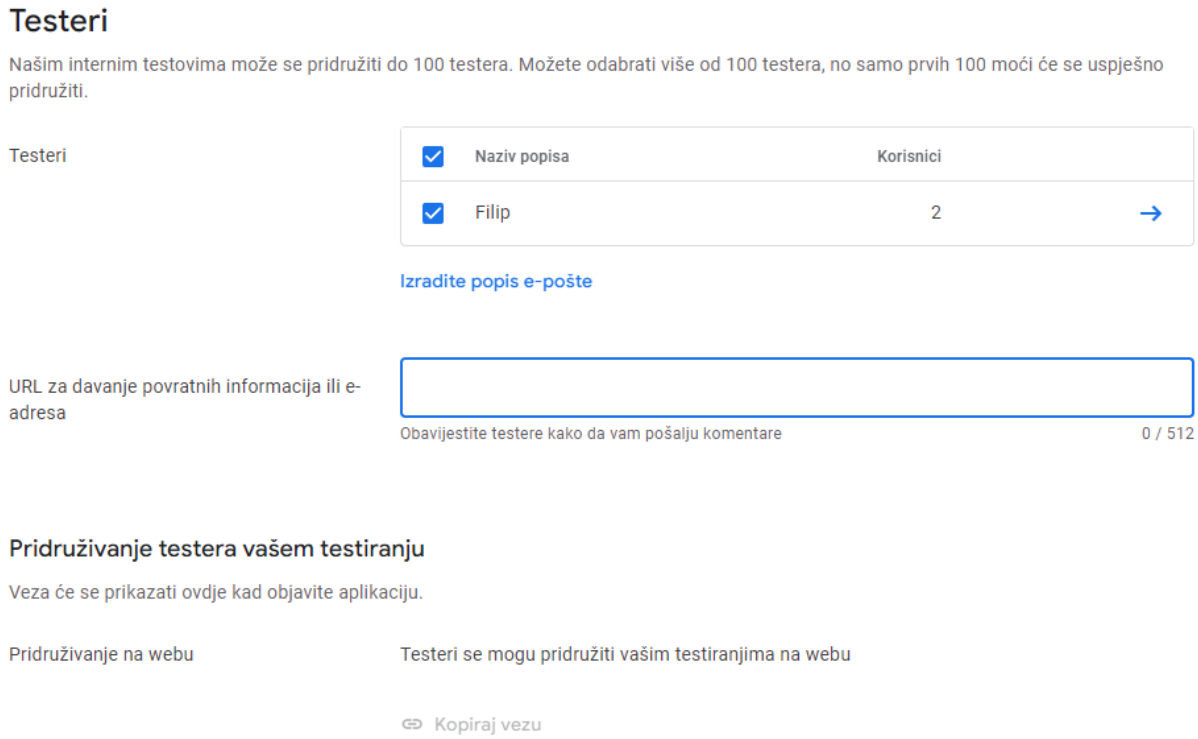
Kako bi testirali unutar aplikacijske kupovine, potrebno je prvo napraviti profil na *Google Play Consoleu* i postaviti aplikaciju tako da već ima pripremljenu logiku za nju. Kod samog kreiranja profila, potrebno je jednokratno platiti 25,00 američkih dolara za aktivaciju. Poslije kreiranja profila, treba odabrati opciju „kreiraj aplikaciju“ (eng. Create app). Prilikom kreiranja aplikacije, potrebno je ispuniti polja prikazana na slici broj 9.



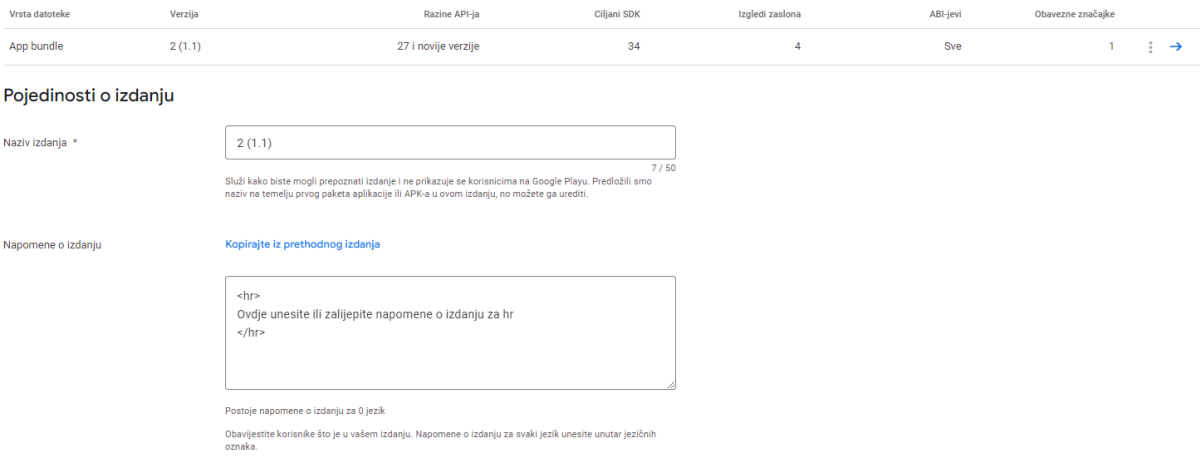

Slika 9: Kreiranje aplikacije u *Google Play Concoleu*

Opciju o tome hoće li aplikacija tražiti plaćanje po instalaciji ili će biti besplatna, može se mijenjati sve dok se aplikacija ne objavi. Kako bi se mogle testirati funkcionalnosti, a pogotovo funkcionalnost unutar aplikacijske kupovine, potrebno je dodati osobe koje će se voditi kao testeri aplikacije.

Na slici 10 prikazana je slika zaslon za dodavanje testera aplikacije, a slika 11 prikazuje zaslon za prijenos *app bundlea* koji mora biti generiran s potpisom. Inače se prikazuje greška.




Slika 10: Dodavanje testera




Slika 11: Prijenos *app bundlea*

Nakon postavljanja aplikacije, potrebno je dodati proizvode koji će se moći kupiti unutar nje i zatim im odrediti naziv i cijenu. Jedna od značajki je prilagođavanje cijene zasebno za svaku državu, što omogućuje kontroliranje i prilagođavanje cijene. Kao što je već napomenuto, za implementaciju unutar aplikacijske kupovine kod korištenja *Google Play Consolea*,

potrebno je koristiti *SDK Google Play Billing*. Kako bi uopće mogli koristiti *Google Play Billing,* neophodno je dodati ovisnost u *build.gradle* datoteku koja je na razini aplikacije (kôd 9) (Android Developers, 2024a).

```
implementation("com.android.billingclient:billing-ktx:7.0.0 ")
```

Kôd 9: Ovisnost za *Google Billing* (Android Developers, 2024a)

Nakon dodavanja ovisnosti i sinkroniziranja promjena, mogu se koristiti klase i funkcije iz biblioteke. Prvi korak je inicijalizacija *BillingClienta* na kojeg se dodaje slušač događaja tipa *PurchasesUpdatedListener* (kôd 10) (Android Developers, 2024a).

```
private val purchasesUpdatedListener =
   PurchasesUpdatedListener { billingResult, purchases -> … }

private var billingClient = BillingClient.newBuilder(context)
   .setListener(purchasesUpdatedListener).build()
```

Kôd 10: Inicijalizacija *BillingClienta* (Android Developers, 2024a)

Sljedeći korak je uspostavljanje veze s *Google Playom* pomoću *startConnection()* metode iz sučelja *BillingClient*. Potrebno je dodati i slušač događaja *BillingClientStateListener* radi obrade rezultata spajanja sa *Google Playom* (Android Developers, 2024a).

```
billingClient.startConnection(object : BillingClientStateListener {
    override fun onBillingSetupFinished(billingResult: BillingResult) {
        if (billingResult.responseCode ==  BillingResponseCode.OK) { … }
    }
    override fun onBillingServiceDisconnected() {}})
```

Kôd 11: Uspostava veze (Android Developers, 2024a)

Potrebno je nadjačati metodu *onBillingSetupFinished()* koja se izvršava kada se dobije rezultat od *Google Play* poslužitelja. Također, treba nadjačati metodu *onBillingServiceDisconnected()* koja se izvršava kada se prekine veza između servera i klijentskog mobilnog telefona (Android Developers, 2024a). Za prikaz proizvoda i lokaliziranih informacija, neophodno je poslati upit uz pomoć *queryProductDetailsAsync()*. Za obradu rezultata, potrebno je dodati slušač događaja koji nasljeđuje *ProductDetailsResponseListener*,

a zatim nadjačati metodu *onProductDetailsResponse()* koja se izvršava kada se dobije rezultat (Android Developers, 2024a).

```
val queryProductDetailsParams =
    QueryProductDetailsParams.newBuilder()
        .setProductList(
            ImmutableList.of(
                Product.newBuilder()
                    .setProductId("product_id_example")
                    .setProductType(ProductType.SUBS)
                    .build()))
        .build()

billingClient.queryProductDetailsAsync(queryProductDetailsParams) {
    billingResult,
    productDetailsList }
```

Kôd 12: Parametri (Android Developers, 2024a)

Parametri za upit prosljeđuju se uz pomoć *QueryProductDetailsParams*, a korištenjem postavljača (eng. *setter*) postavlja se identifikator koji je dodijeljen proizvodu u *Google Play Consoleu*. Osim toga, može se dohvaćati tip proizvoda putem *setProductType* koji postavlja tip proizvoda kojeg treba dohvatiti (Android Developers, 2024a). Proizvodi i informacije o njima, mogu se dohvatiti i bez funkcija povratnih poziva ako se koristi *queryProductDetails()* koji se poziva unutar korutine. Implementacija takvog načina zahtjeva dodavanje ovisnosti za korutine unutar *build.gradle* datoteke koja je na razini aplikacije (kôd 13) (Android Developers, 2024a), a implementacija je prikazana u kôdu 14.

```
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")
```

Kôd 13: Ovisnost za *korutine* (Android Developers, 2024)

```
suspend fun processPurchases() {
    val productList = listOf(
        QueryProductDetailsParams.Product.newBuilder()
            .setProductId("product_id_example")
            .setProductType(BillingClient.ProductType.SUBS)
            .build())
    val params = QueryProductDetailsParams.newBuilder()
    params.setProductList(productList)
```

```
    val productDetailsResult = withContext(Dispatchers.IO) {
        billingClient.queryProductDetails(params.build())}}
```

Kôd 14: Implementacija bez funkcija povratnih poziva (Android Developers, 2024a)

Kao što je napomenuto, ne koriste se funkcije povratnog poziva nego je pozvana funkcija unutar korutine s opcijom ulaz-izlaz (eng. Input-Output). Navedenu opciju koriste korutine koje se bave ulazno izlaznim podacima, kao što su mrežni upiti, čitanje i pisanje datoteka ili interakcija s bazom podataka. Ovakav način slanja upita omogućuje veću čitljivost koda (Android Developers, 2024a). Dobiveni podaci nalaze se u listi objekata tipa *ProductDetails*, koja ima nekoliko atributa kojima se pristupa pomoću metoda za dohvat podataka (eng. *getter*), na primjer *getDescription()*, *getName()* i *getProductType()* (Android Developers, 2024a).

Za implementaciju kupnje proizvoda, koristi se *launchBillingFlow()* metoda sučelja *BillingClienta,* koja za parametre prima aktivnost i klasu *BillingFlowParams*. Navedenoj klasi postavlja se proizvod koji se želi kupiti. *launchBillingFlow()* vraća odgovor, a kod *OK* označava uspjeh (kôd 15) (Android Developers, 2024a). U varijablu *productDetailsParams* spremaju se proizvodi koji se žale prikazati na ekranu za kupnju (kôd 16).

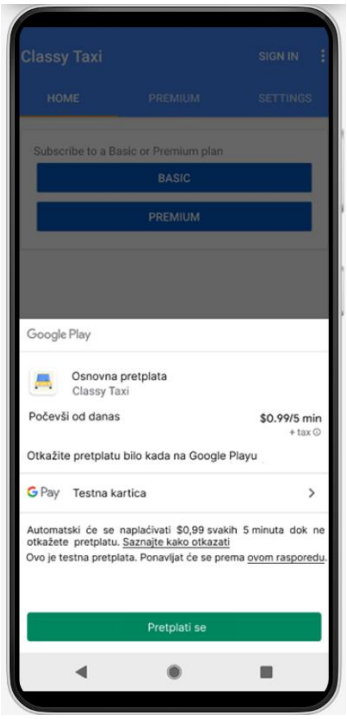
```
val billingFlowParams = BillingFlowParams.newBuilder()
    .setProductDetailsParamsList(productDetailsParamsList)
    .build()

val billingResult = billingClient.launchBillingFlow(activity, billingFlowParams)
```

Kôd 15: Kod za kupnju proizvoda (Android Developers, 2024a)

```
val productDetailsParamsList = listOf(
    BillingFlowParams.ProductDetailsParams.newBuilder()
queryProductDetailsAsync().setProductDetails(productDetails)
        .setOfferToken(selectedOfferToken).build())
```

Kôd 16: Lista detalja (Android Developers, 2024a)

Bitno je napomenuti da je za supskripcije potreban i *offer token* koji je moguće dobiti pozivom funkcije *ProductDetails.subscriptionOfferDetails()* kako bi se prikazale sve supskripcije koje su dostupne za kupnju, a za proizvod koji se kupuje samo jednom nije potrebno pozvati metodu *setOfferToken()* (Android Developers, 2024a).




Slika 12: *Billing flow* (Android Developers, 2024a)

Prilikom pozivanja funkcije *launchBillingFlow()* otvara se zaslon koji omogućuje kupnju proizvoda koji je proslijeđen toj funkciji. Nakon klika na gumb *Subscribe*, *Google Play* poziva metodu *onPurchasesUpdated()* radi vraćanja rezultata. Da bi se rezultat mogao obraditi, potrebno je postaviti slušač događaja koji nasljeđuje *PurchasesUpdatedListener*, a on je postavljen prilikom inicijalizacije *BillingClienta* (Android Developers, 2024a).

```
override fun onPurchasesUpdated(billingResult: BillingResult, purchases:
List<Purchase>?) {
   if (billingResult.responseCode == BillingResponseCode.OK && purchases != null) {
       for (purchase in purchases) {
           handlePurchase(purchase)
       }
         } else if (billingResult.responseCode == BillingResponseCode.USER_CANCELED)
{} else {}}
```

Kôd 17: *OnPurchaseUpdated()* (Android Developers, 2024a)

Ako je vraćeni kod *OK* (*HTTP 200*), može se rukovati kupnjom. Drugi kod koji može vratiti *launchBillingFlow()* je *USER_CANCELED*, a on se vraća kada korisnik odluči prekinuti kupnju proizvoda. Nakon uspješne kupnje, prikazuje se zaslon koji označava uspješnu kupnju. Također, prilikom svake kupnje generira se jedinstveni token. On služi za označavanje

korisnika i proizvoda kojeg je korisnik kupio. Također, korisniku se nakon uspješne kupnje šalje poruka na elektroničku poštu koja sadrži račun i jedinstveni identifikator kupnje (Android Developers, 2024a).

Nakon završetka kupnje, potrebno je verificirati kupnju i „predati“ proizvod kupcu, te označiti da kupac posjeduje kupljeni proizvod, ako je status transakcije *PURCHASED*. Verificiranje kupnje moguće je napraviti na nekoliko načina, a sve ovisi o aplikaciji koja se razvija. Ako aplikacija omogućuje transakcije s dva ili više koraka, stanje transakcije može imati dva statusa, *PENDING* i *PURCHASED*, a status se može dobiti pozivanjem metode *getPurchaseStatus()*. Transakcije sa statusom *PENDING* moraju biti odobrene prilikom inicijalizacije *BillingClienta* pozivanjem metode *enablePendingPurchases()* (Android Developers, 2024a).

#### 3.2.3.1. Potvrda kupnje jednokratnih proizvoda i pretplata

Potvrđivanje kupnje jednokratnih proizvoda kod korištenja *Google Play Billing Librarya* može se učiniti na dva načina. Prvi način je potvrda kupnje na poslužiteljskom dijelu aplikacije (eng. *backend*), a drugi način je potvrđivanje unutar mobilne aplikacije (Android Developers, 2024a). Sljedećim kodom prikazano je potvrđivanje kupnje unutar aplikacije za mobilne uređaje.

```
val client: BillingClient = ...
val acknowledgePurchaseResponseListener: AcknowledgePurchaseResponseListener = ...

suspend fun handlePurchase() {
    if (purchase.purchaseState === PurchaseState.PURCHASED) {
        if (!purchase.isAcknowledged) {
            val acknowledgePurchaseParams = AcknowledgePurchaseParams.newBuilder()
                    .setPurchaseToken(purchase.purchaseToken)
            val ackPurchaseResult = withContext(Dispatchers.IO) {
               client.acknowledgePurchase(acknowledgePurchaseParams.build())
            }}}}
```

Kôd 18: Potvrda kupnje (Android Developers, 2024a)

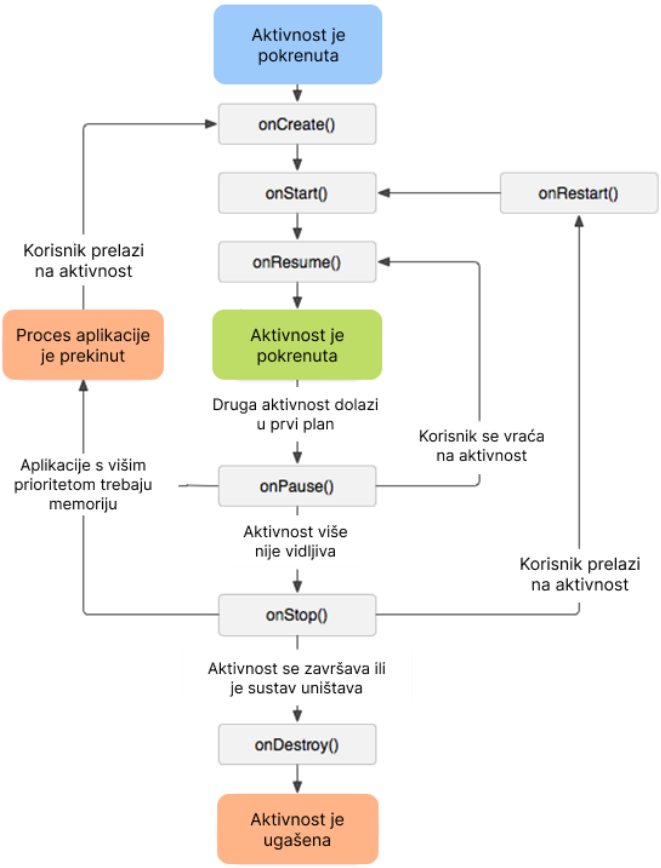
Vidljivo u kodu iznad, prvo se provjerava stanje kupnje korištenjem *purchaseState*, a tek nakon što je stanje kupovine jednako vrijednosti kupljeno (eng. *PURCHASED*), priznaje se kupnja proizvoda (Android Developers, 2024a). Priznavanje kupnje pretplata radi se na sličan način kao i priznavanje kupnje jednokratnih proizvoda. Razlika je u tome što je kod priznavanja

kupnje pretplata potrebno napraviti samo prilikom prve kupnje, a obnavljanje supskripcija nije potrebno ponovo priznavati (Android Developers, 2024a).

#### 3.2.3.2. Ažuriranje stanja kupovine

Kao što je navedeno na stranici za implementaciju *Google Play Billing Librarya*, nije dovoljno pratiti kupnje samo putem slušača događaja *PurchasesUpdatedListener*, zato što postoji mogućnost da aplikacija „izgubi trag“ kupnje (Android Developers, 2024a). Prvi scenarij u kojem se može „izgubiti trag“ o kupnji proizvoda, odnosi se na situaciju kada korisnik obavlja kupovinu prilikom problema s mrežom. Na primjer, korisnik obavi kupovinu te primi potvrdu od *Googlea*, ali uređaj izgubi mrežnu povezanost prije primanja obavijesti o kupnji putem *PurchasesUpdatedListenera* (Android Developers, 2024a). Drugi scenarij odnosi se na korištenje više uređaja od strane korisnika. Prilikom toga, nije dovoljno priznati kupnju proizvoda samo na uređaju kojim je korisnik obavio kupnju. Na primjer, korisnik obavi kupnju na jednom uređaju, a zatim koristi drugi uređaj na kojem mora imati mogućnost korištenja kupljenog proizvoda. Drugim riječima, proizvod mora biti vezan na korisnika, a ne na mobilni uređaj (Android Developers, 2024a).

Da bi se riješili problemi koji su navedeni, potrebno je pozvati *queryPurchasesAsync()* nad *BillingClientom* u *onResume()* funkciji (Android Developers, 2024a), što je prikazano na slici 13.




Slika 13: Životne funkcije aktivnosti u aplikaciji za *Android OS* (Android Developers, 2024a)

Iščitano iz grafa iznad, funkcija *onResume()* je funkcija životnog vijeka aktivnosti koja se pokreće prilikom vraćanja aktivnosti u fokus. Također, poziva se kada aktivnost prelazi iz pauziranog (eng. paused) stanja u aktivno (eng. active) stanje. Time se osigurava da aplikacija ima zadnje stanje kupljenih proizvoda, neovisno o problemima s mrežom i korištenjem više uređaja (Android Developers, 2024a).

Kao što je navedeno ranije, moguće je mijenjati cijenu proizvoda prema državama. Da bi se prikazala ispravna cijena za državu koju korisnik koristi u *Google Playu*, potrebno je pozvati metodu *getBillingConfigAsync()* nad klasom *BillingClient* (Android Developers, 2024a). Sljedeći kod omogućuje prikaz prilagođene cijene prema državi korisnika.

```
val getBillingConfigParams = GetBillingConfigParams.newBuilder().build()
billingClient.getBillingConfigAsync(getBillingConfigParams,
    object : BillingConfigResponseListener {
        override fun onBillingConfigResponse(
            billingResult: BillingResult,
            billingConfig: BillingConfig?
        ) {
            if (billingResult.responseCode == BillingResponseCode.OK
```

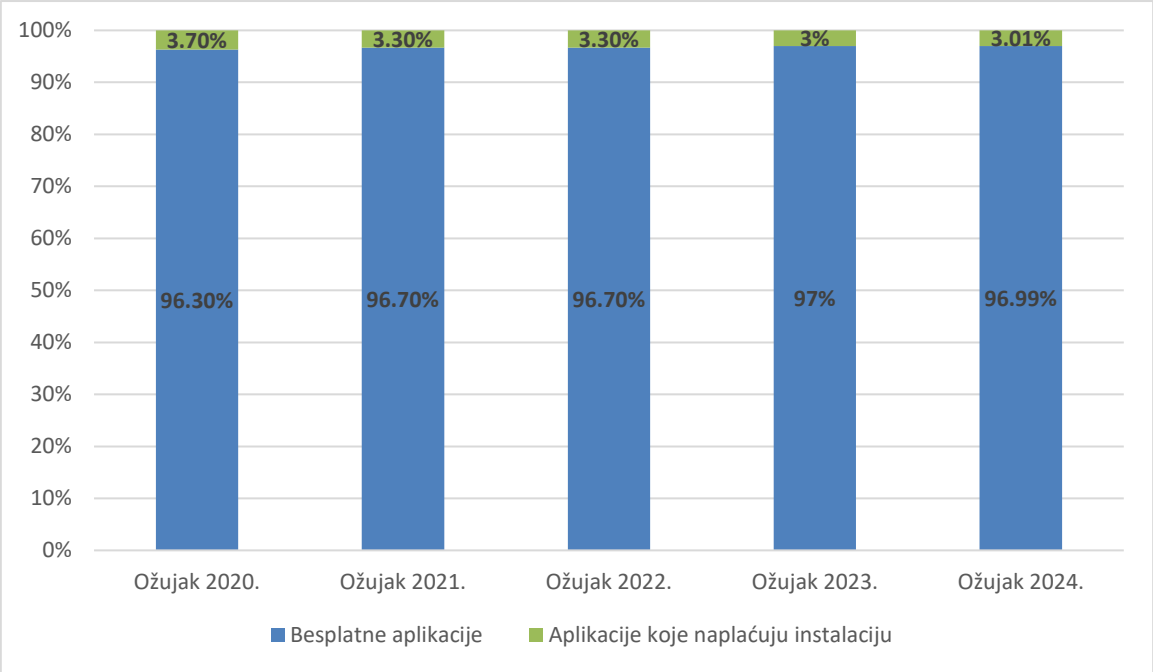
```
                && billingConfig != null) {
                val countryCode = billingConfig.countryCode
            } else {}}
})
```

Kôd 19: Prikaz prilagođene cijene (Android Developers, 2024a)

U kodu iznad vidljivo je da metoda *getBillingConfigAsync()* prima *GetBillingConfigParams* i *BillingConfigResponseListener* kako bi se dobio rezultat. Radi se provjera statusa odgovora, na temelju čega se može iščitati kôd države koju je korisnik postavio u *Google Playu* (Android Developers, 2024a).

## 3.3. Plaćena ili besplatna instalacija

Jedno od najvećih pitanja i nedoumica kod izdavača jest da li aplikacija koja se izdaje treba zahtijevati plaćanje po instalaciji ili je bolje da je aplikacija besplatna za preuzimanje. Na sljedećoj slici prikazan je udio aplikacija koje ne naplaćuju instalaciju naspram aplikacija koje naplaćuju instalaciju.




Slika 14: Besplatne vs. plaćene instalacije (Ceci, 2024.)

Od 2020. do 2024. godine, udio aplikacija za *Android* operacijski sustav koje zahtijevaju plaćanje instalacije smanjuje se. Razlika u broju aplikacija koje zahtijevaju plaćanje instalacije je manja za 31.800 u ožujku 2023. godine u odnosu na listopad 2019. godine (Ceci, 2024).

### 3.3.1. Prednosti i nedostaci besplatne instalacije

Prednost besplatnih instalacija aplikacije je doseg korisnika. Naime, korisnici će uvijek prvo pregledati besplatne opcije aplikacija koje su dostupne, te samim time, besplatna instalacija aplikacije dobar je izbor (Genadinik, 2014). Druga prednost besplatnih instalacija aplikacije je da se monetizacija može napraviti na drugačiji način, kao što je prikaz reklama unutar aplikacije ili unutar aplikacijska kupovina. Treća prednost nadovezuje se na prvu. Više korisnika će preuzeti aplikaciju i imat će manja očekivanja, a samim time recenzije aplikacije biti će bolje (Genadinik, 2014).

Najveći problem kod besplatne instalacije je da izdavači nemaju siguran prihod od aplikacije, pogotovo ako je odlučeno da će monetizacija aplikacije biti odrađena pomoću unutar aplikacijskih kupovina i pretplata, ali bez reklama (Genadinik, 2014). Drugi problem veže se na prve dvije navedene prednosti, jer iako će više korisnika instalirati aplikaciju, to ne znači da će korisnici nastaviti koristiti aplikaciju. Budući da je aplikacija besplatna za preuzimanje, korisnici će često skinuti aplikaciju iz radoznalosti i onda ju deinstalirati. Time se smanjuje prihod od unutar aplikacijskih kupovina i reklama (Genadinik, 2014).

### 3.3.2. Prednosti i nedostaci plaćene instalacije

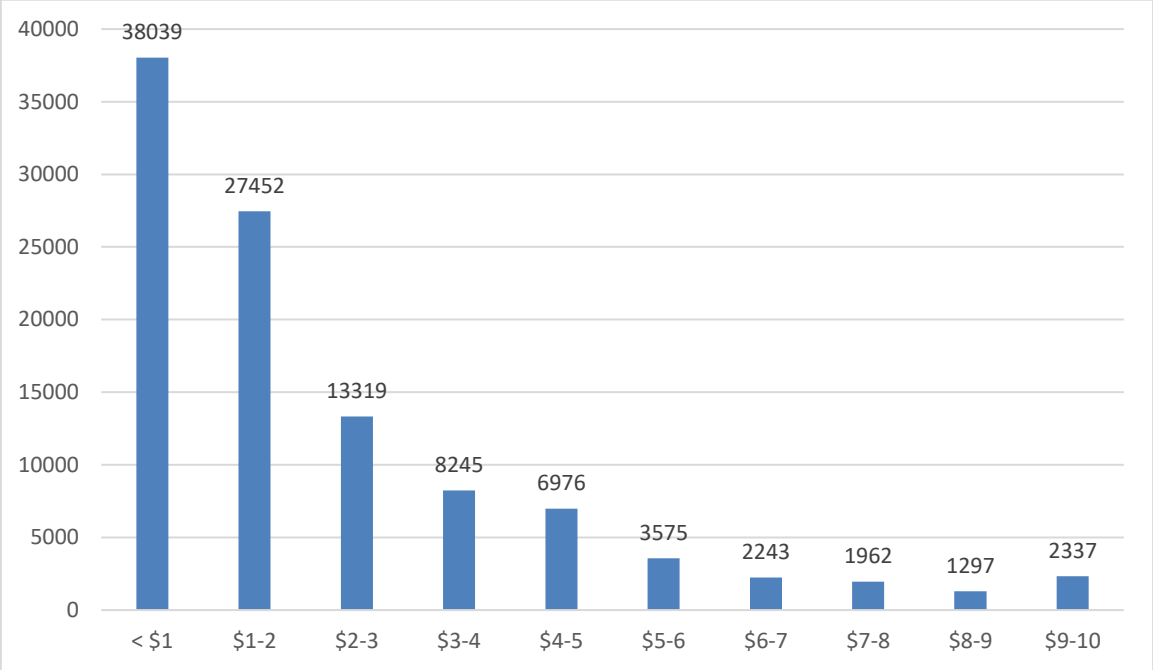
Prednost naplaćivanja instalacije je da izdavači imaju siguran prihod od svakog korisnika koji će se odlučiti instalirati aplikaciju (Genadinik, 2014). Druga prednost je da će korisnici koji su platili instalaciju biti vjerniji i koristit će ju duže od besplatnih aplikacija jer su platili aplikaciju (Srinidhi, 2020).

Sve prednosti kod besplatne instalacije su nedostaci plaćene instalacije. Kao što je navedeno, korisnici će uvijek prvo istražiti besplatne opcije, a tek tada opcije koje se moraju plaćati (Genadinik, 2014). Dodatna mana plaćenih instalacija je da će korisnici očekivati aplikaciju koja ne prikazuje reklame ili nema puno unutar aplikacijskih kupovina koje poboljšavaju korisničko iskustvo. Odnosno, korisnici očekuju da je aplikacija potpuna, budući da su morali platiti instalaciju (Srinidhi, 2020).

### 3.3.3. Kada i koliko naplaćivati instalaciju

Naplaćivanje instalacije aplikacije može biti jako uspješno kod aplikacija koje mogu biti sponzorirane od poznatih ličnosti ili još bolje, ako je izdavatelj poznata ličnost. Sponzorstvo daje sigurnost da će korisnici instalirati aplikaciju, jer će vidjeti da ju njihovi idoli koriste ili će ju poznate ličnosti reklamirati na svojim *You Tube* kanalima i ostalim društvenim mrežama. Time se osigurava da aplikacija ima dovoljan doseg (Genadinik, 2014). Instalacija se može naplaćivati za aplikacije koje su dovoljno specifične, s obzirom na to da ne postoje ili postoje

vrlo rijetke besplatne alternative koje pružaju jednako kvalitetno rješenje. Drugim riječima, instalacija se može naplaćivati ako nema puno konkurencije na tržištu (Genadinik, 2014). Ako aplikacija spada pod kategorije koje su ranije navedene ili je odlučeno da će se naplaćivati instalacija aplikacije, unatoč konkurenciji i njenoj domeni, potrebno je donijeti odluku o cijeni instalacije. Slika 15 prikazuje raspon cijene.




Slika 15: Broj aplikacija s obzirom na cijenu instalacije (Enterprise Apps Today, 2023)

Na grafu prikazanom na slici 15, može se zaključiti da većina aplikacija koje traže plaćanje prije instalacije, ima cijenu manju od 1,00 američkog dolara. Kako navodi Genadinik (2014), postoje tri kategorije raspona cijena za mobilne aplikacije. Prvi raspon je 0,99 – 1,99 američkih dolara i smatra se najnižim rasponom cijene. Drugi raspon je 1,99 – 2,99 američkih dolara, a treći 2,99 – 4,99 američkih dolara. Ako se pogleda graf i pripadajući raspon cijena prema Genadiniku (2014), može se uočiti da bi se najniži rang trebao proširiti prema 0,00 američkih dolara tako da je prvi raspon od 0,50 do 1,99 američkih dolara. Prema navedenom, cijena instalacije aplikacije trebala bi biti od 1,00 – 3,00 američkih dolara, no ako aplikacija nudi nešto što konkurencija ne pokriva, može se postaviti viša cijena instalacije. Kao još jedan parametar, može se uzeti i mogućnost određivanja cijene aplikacije prema zemlji korisnika kao što omogućuje *Google Console*. Ova funkcionalnost je vrlo dobra jer se cijena može prilagoditi stanju svake pojedine države. Time će korisnici iz tih država češće preuzeti aplikaciju koja naplaćuje instalaciju, ukoliko je cijena prilagođena njima i njihovoj državi.

## 3.4. *Freemium* model mobilnih aplikacija

Definicija *freemiuma* je „način naplate proizvoda ili usluge u kojem je osnovni proizvod ili usluga besplatna, ali kupac plaća dodatne značajke“ (Cambridge, 2024). Druga definicija je „kombinacija riječi besplatno i premium, freemium je vrsta poslovnog modela koji nudi osnovne značajke proizvoda ili usluge korisnicima bez ikakvih troškova i naplaćuje za dodatke ili napredne značajke“ (Segal, 2024), dok Alexander Osterwalder i Yves Pigneur *freemium* definiraju kao „poslovni model, uglavnom temeljeni na webu, koji stapa slobodne usluge s plaćenim uslugama“ (Osterwalder & Pigneur, 2010).

Sve navedene definicije spominju besplatne osnovne značajke i mogućnost korisnika da plati dodatne značajke. Moglo bi se reći kako *freemium* „spaja“ prednosti aplikacija koje su besplatne i onih koje naplaćuju instalaciju. Naime, aplikacija će imati jako dobar dohvat korisnika (besplatna je), ali za one korisnike kojima su potrebne dodatne značajke koje se nude u *premium* verziji, aplikacija naplaćuje „otključavanje“ tih značajki (plaćena instalacija).

Iz ovih definicija mogu se izvesti četiri ključne točke *freemium* modela:

- Aplikacija je besplatna za preuzimanje
- Osnovna aplikacija i osnovne funkcionalnosti moraju biti besplatne
- Aplikacija nudi dodatne značajke za korisnike
- Korištenje dodatnih značajki se naplaćuje

Prema navedenim točkama, aplikacije kojih se ljudi prvo sjete ne spadaju pod *freemium* aplikacije. Često se može zavarati time, jer su neke aplikacije besplatne za preuzimanje i automatski pomislimo da su bazirane na *freemium* modelu. Za primjer se mogu uzeti mobilne aplikacije kao što su *Dropbox* i *Netflix*. Kako bi vidjeli jesu li ove aplikacije bazirane na *freemium* modelu, mogu se osvrnuti na četiri ključne točke ranije navedene. U sljedećem primjeru, provjerava se slijede li aplikacije *freemium* model.

- *Netflix*: mobilna aplikacija je besplatna za preuzimanje
- *Netflix*: korisnici NE MOGU koristiti niti jednu značajku aplikacije ako nisu pretplaćeni
- *Netflix*: aplikacija ima ponudu dodatnih značajki (više profila, bolja kvaliteta)

- *Netflix*: korisnici dodatne značajke moraju plaćati
- *Dropbox*: aplikacija je besplatna za preuzimanje
- *Dropbox*: korisnici mogu koristiti aplikaciju bez plaćanja
- *Dropbox*: aplikacija nudi dodatne značajke
- *Dropbox*: korisnici moraju platiti da bi koristili dodatne značajke

Na prvi pogled može se smatrati kako *Netflix* aplikacija slijedi *freemium* model, jer je besplatna za preuzimanje, a za korištenje se mora preplatiti. Upravo pretplata daje do znanja da aplikacija ne slijedi model. Naime, korisnici ne mogu koristiti aplikaciju ako se ne odluče za pretplatu. S druge strane, *Dropbox* slijedi *freemium* model zato što je besplatan za preuzimanje, a korisnici mogu koristiti aplikaciju bez da kupuju unutar nje. Također, nude se dodatne značajke (više prostora za pohranu podataka) za određenu cijenu.

Nadalje, primjer aplikacije koja ne slijedi *freemium* model je *Nova Launcher* i *Nova Launcher Prime*, a razlog je instalacija druge aplikacije. Naime, kako bi se otključale dodatne značajke potrebno je instalirati drugu aplikaciju. Dakle, da bi se smatralo da aplikacija slijedi *freemium* model, otključavanje dodatnih značajki mora se odraditi u istoj aplikaciji u kojoj je korisnik koristio osnovne, besplatne značajke.

Prema Osterwalderu i Pigneuru, za *freemium* model najvažnija je platforma, jer ona mora omogućiti pružanje besplatnih osnovnih usluga po niskim marginalnim troškovima. Troškovna struktura ima tri dijela: značajni fiksni troškovi, niski marginalni troškovi za usluge korisnika koji imaju besplatne račune i troškovi za korisnike sa *premium* računima. Isto tako, definiraju da odnosi s korisnicima moraju biti automatizirani i jednostavni kako bi se moglo upravljati velikim brojem besplatnih korisnika (Osterwalder & Pigneur, 2010). Kao ključne pokazatelje, Osterwalder i Pigneur naveli su stopu konverzije besplatnih računa u *premium* račune, broj korisnika koje je moguće privući i troškove stjecanja korisnika i pružanja usluga (Osterwalder & Pigneur, 2010).

### 3.4.1. Prednosti *Freemium* modela

Prema *Segalu* postoje tri prednosti modela:

1. Lako privlačenje potencijalnih korisnika i skupljanje podataka (Segal, 2024)
2. Prihodi putem oglasa (Segal, 2024)

3. Velika prepoznatljivost brenda (za *startupe*) (Segal, 2024)

Važno je napomenuti da se prihodi putem oglasa, u većini slučajeva, odnose na oglase koji se prikazuju korisnicima sa besplatnim profilima. To je vrlo bitno jer korisnici koji su platili *premium* verziju aplikacije očekuju da ona nema oglasa koji će narušiti korisničko iskustvo. O prikazivanju oglasa u *premium* verziji aplikacije može se zaključiti kako se oni ne bi trebali prikazivati, osim ukoliko aplikacija nudi posebnu verziju ili značajku koja uklanja reklame. *Business Model Toolbox* naveo je prednosti koje su povezane sa prvom prednosti koju je naveo *Segal*:

1. „Učinak na marketing: usmena predaja radi sklonosti ljudi o širenju informacija o besplatnim alatima i uslugama“ (Business Model Toolbox, 2024)
2. „Mrežni učinak: ako puno ljudi koristi uslugu, veća je vjerojatnost da će usluga privući druge korisnike“ (Business Model Toolbox, 2024)

Osim prethodno navedenih prednosti, može se zaključiti da je *freemium* model vrlo dobra opcija za *startupove*, ali ne samo radi prepoznatljivosti brenda, nego i radi relativno niskog početnog troška. Ovaj zaključak je potvrđen time da se *MVP* (eng. *Minimum Viable Product*) može iskoristiti kao besplatna verzija aplikacije. Navedena će verzija dobiti povratne informacije na kojima se može graditi *premium* verzija aplikacije. Time se dobiva vrlo dobra *premium* verzija koja radi ono što korisnici zapravo žele i smatraju da im je potrebno. Uz to, radi povratnih informacija korisnika, može se svakako predočiti da će stopa konverzije korisnika koji imaju besplatne profile i onih koji imaju *premium* profile biti bolja, a posljedično će i prihod biti veći.

### 3.4.2. Nedostaci *Freemium* modela

Kao nedostatke prema Business Model Toolbox (2024), Zaki (2024) i Segal (2024), mogu se navesti:

1. Tanka linija između održavanja besplatnih korisnika i davanja previše besplatnih značajki
2. Previše oglasa može jako smetati besplatnim korisnicima te će u potpunosti prestati koristiti aplikaciju

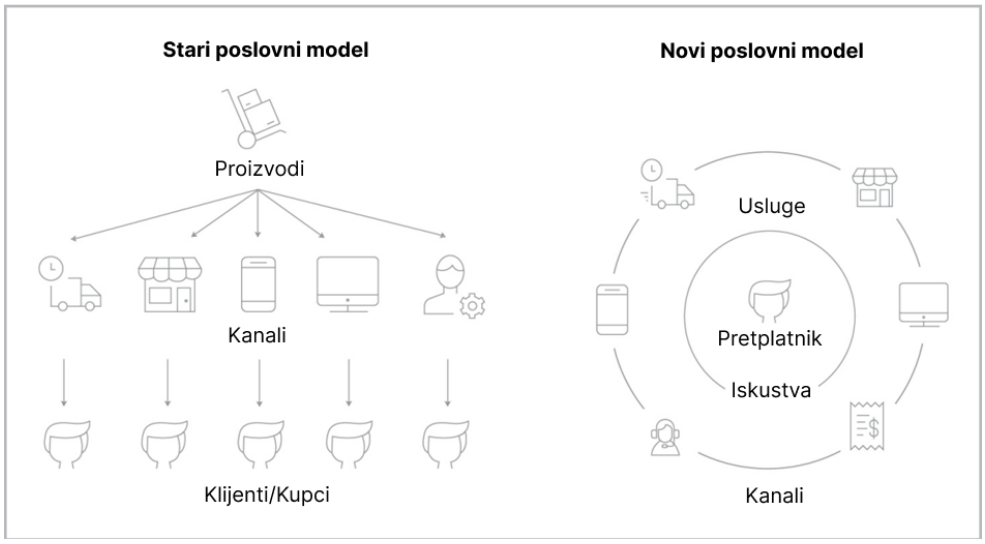
3. Ne može se osigurati dobra konverzija korisnika

Navedeni nedostaci pokrivaju sve aspekte, no može se napomenuti kako prva točka najviše utječe na *startupe*. Naime, *startupi* nemaju velike prihode pa ukoliko se odluče za *freemium* model, vrlo lako mogu ostati bez sredstava za nastavak razvijanja aplikacije, ali i razvijanja i napretka samog *startupa*.

## 3.5. Pretplatnički model

U današnje vrijeme, sve više aplikacija koristi pretplatnički model. Definicijom danom od Tarvera (2024), pretplatnički model objašnjava se kao „model koji se bazira na ideji prodaje proizvoda ili usluge kako bi ostvario prihod od mjesečne ili godišnje pretplate“. Pretplata (eng. *subscription*) sama po sebi traži ponavljajuće plaćanje od strane korisnika. To plaćanje može biti na tjednoj, mjesečnoj, godišnjoj ili nekoj proizvoljnoj bazi. Kada se pogleda definicija, može se uočiti da pretplatnički model ima jako veliku prednost, a to je – ponavljajući prihod od pretplate korisnika.

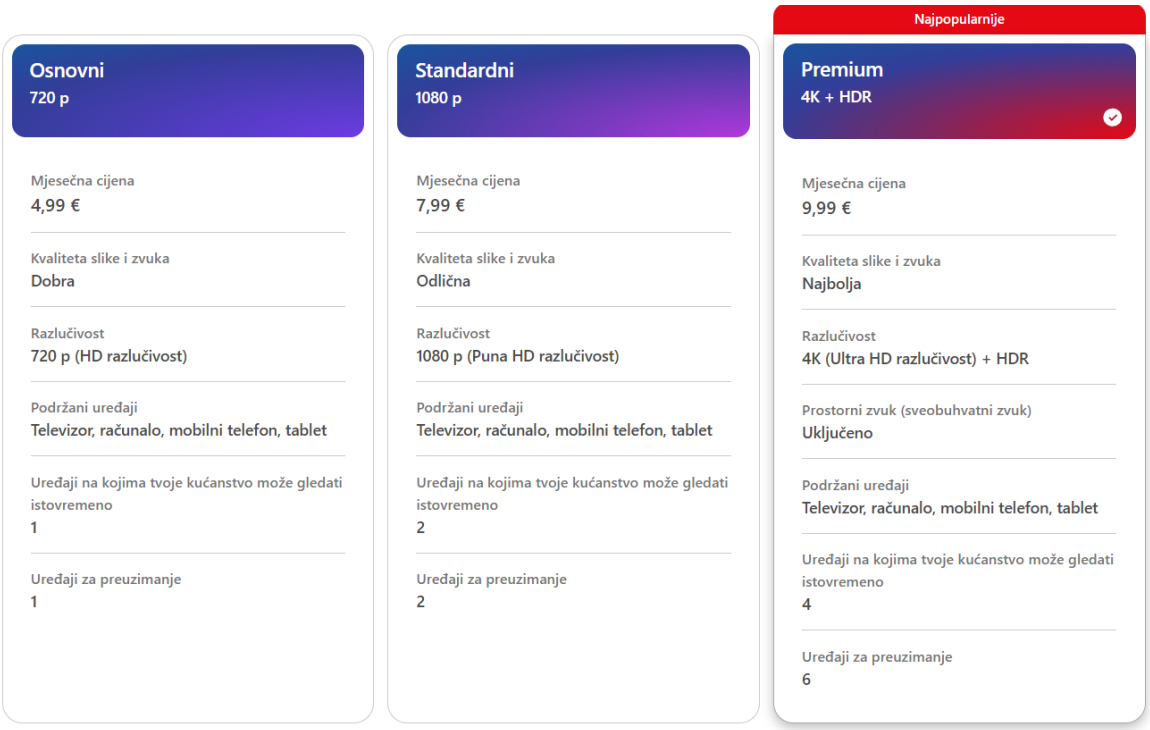
Kao što navode Tzuo & Weisert, pretplata je bila korištena i osamdesetih godina dvadesetog stoljeća, a kao primjer navode pretplatu na *CD* izdanja pjesama i albuma (Tzuo & Weisert, 2018). Time se potvrđuje da pretplata i pretplatnički poslovni model nisu novost u današnjem svijetu. Tzuo & Weisert naveli su dvije ključne točke kojima objašnjavaju zašto je pretplatnički model dobio na popularnosti. Prva je da se danas pretplate dostavljaju digitalnim putem pa je radi toga lakše skalirati sustave i infrastrukturu u odnosu na broj korisnika. Drugu točku naveli su kao izmjenu fokusa i razmišljanja organizacija (Tzuo & Weisert, 2018). Na slici u nastavku prikazana je razlika između starog i novog razmišljanja i fokusa organizacija.




Slika 16: Stari vs. novi poslovni model (Tzuo & Weisert, 2018)

Tzuo & Weisert kao najbitniju stavku iz knjige, naveli su dijagram prikazan na slici 16. Naime, dijagram prikazuje kako se promijenio fokus organizacija i samim time njihov poslovni model. Stari model fokusiran je na prodaju čim više jedinica proizvoda, koji su bili distribuirani različitim linearnim kanalima. Današnje organizacije su se promijenile i u fokus stavljaju kupca. Na desnoj strani dijagrama, vidi se da su kanali cirkularni. Tzuo & Weisert govore kako je to u sklopu sa time što današnje organizacije shvaćaju da korisnike mogu pronaći putem više kanala. Na taj način organizacije mogu saznati više informacija o korisniku, a čim više znaju o korisniku tim ga bolje mogu poslužiti. Zapravo, kako Tzuo & Weisert navode, „to je digitalna transformacija: od linearnih transakcijskih kanala do kružnog, dinamičnog odnosa s korisnicima/pretplatnicima“ (Tzuo & Weisert, 2018). Također, ističu da je „promjena od produktno usmjerenog do korisnički usmjerenog organizacijskog načina razmišljanja, ključna karakteristika ekonomije pretplate“ (Tzuo & Weisert, 2018).

Kao primjer aplikacije koja koristi čisti pretplatnički model, može se navesti *Netflix* aplikacija. Ona koristi čisti pretplatnički model i ako je korisnik želi koristiti, nužno je kupiti pretplatu i to raditi na mjesečnoj bazi. Na sljedećoj slici dan je prikaz pretplatničkih paketa koje *Netflix* ima u ponudi.




Slika 17: *Netflix* pretplatnički paketi (Netflix, 2024)

Na slici 17, vidljivo je da prvi paket ima najlošiju razlučivost i kvalitetu. Ostali paketi nude bolje opcije, ali zato su cjenovno manje pristupačni. Odnosno, *Netflix* ima samo pretplatu kojom održavaju konstantne mjesečne prihode. Uz to, nude različite pakete koji omogućuju korisnicima da sami odrede paket koji je idealan za njih. Organizacije koje koriste pretplatnički model za svoje aplikacije iznajmljuju svoje aplikacije korisniku. Drugim riječima, korisnicima daju softver kao uslugu (eng. *Software as a Service*, dalje korišteno *SaaS*).

### 3.5.1. Prednosti pretplatničkog modela

Prednosti pretplatničkog modela nisu brojne, ali su jako dobre ponajviše radi prve točke:

- Prihod bez velikih fluktuacija (Tarver, 2024)
- Korisnici više vrijednosti (Tarver, 2024)

Može se navesti da je velika prednost pretplatničkog modela zadržavanje korisnika (iako samo po sebi pripada i prvoj i drugoj točki). Dodatno, automatski obnovljive pretplate pospješuju zadržavanje korisnika, budući da nije potrebno ručno plaćanje pretplate.

### 3.5.2. Nedostaci pretplatničkog modela

S druge strane, nedostatke pretplatničkog modela najviše će osjetiti *startupi* i organizacije koje se pokušavaju probiti u novo područje, a oni su:

- Generiranje pretplatnika
- Nezadovoljstvo ponudom
- Postavljanje i promjena cijena

Generiranje pretplatnika teško je za aplikacije koje imaju veći broj konkurencije u svojoj domeni, jer će korisnici već koristiti neki drugi proizvod. Nezadovoljstvo ponudom je nedostatak koji se može pojaviti kod *Netflix* aplikacije. Naime, kada bi *Netflix* maknuo stari, ili ne dodao novi materijal (serije i filmove), neki korisnici bili bi nezadovoljni ponudom, a moglo bi doći do prekida pretplate. Postavljanje i promjena cijena pretplate svakako je vrlo važno kod pretplatničkog modela jer se, kao kod *Netflixa*, paketima za pretplatu mora pokriti vrlo širok raspon korisnika. Mijenjanje cijene je još osjetljivije, jer najčešće ne dolazi samo po sebi (ako ima više paketa), nego se mijenja i ponuda funkcionalnosti ili značajki. To je loše jer može dovesti do manjeg broja korisnika koji će obnoviti pretplatu. Svi navedeni nedostaci mogu se riješiti dobrim marketingom i raznovrsnom ponudom, a navedeno će pomoći u generiranju pretplatnika, smanjiti nezadovoljstvo korisnika i regenerirati one korisnike koji su potencijalno odustali od pretplate nakon promjene cijena.

## 3.6. Monetizacija *B2B* aplikacija

*B2B* (eng. *Business to business*) je poslovni model koji podrazumijeva transakcije između poduzeća, pri čemu jedno poduzeće kupuje proizvode ili se koristi uslugama drugog poduzeća, a time pridonosi ostvarivanju ciljeva organizacije (Seebacher, 2021, str. 192). S druge strane, *B2C* (eng. *Business to consumer*) je poslovni model koji se odnosi na transakcije između poduzeća i krajnjeg korisnika, koji proizvod ili uslugu koristi za osobnu upotrebu. *B2B* aplikacije mogu slijediti pretplatnički model, koji je ranije objašnjen. Drugim riječima, poduzeća mogu plaćati pretplatu na vremenskoj bazi kako bi mogli koristiti aplikaciju.

Fiksna pretplata - poduzeća mogu naplaćivati fiksnu cijenu neovisno o tome koliko korisnika koristi softver. Takav način monetizacije koristi se kada su troškovi izrade predvidljivi. Fiksna cijena eliminira potrebu za pregovorima, što pojednostavljuje kupovinu (BIGCOMMERCE, 2024).

Cijena prema upotrebi - poduzeće naplaćuje cijenu prema potrošnji ili dodatnim troškovima koji mogu biti uzrokovani većom upotrebom podrške (BIGCOMMERCE, 2024).

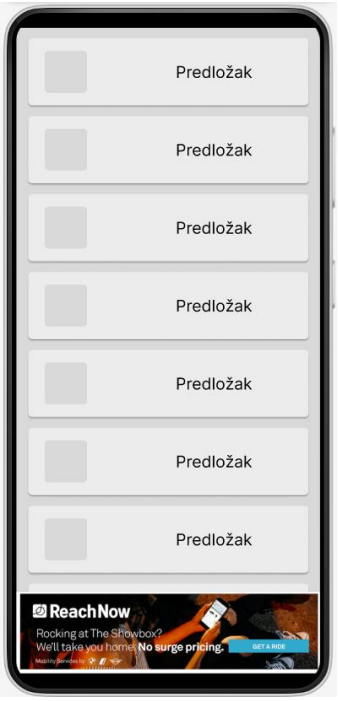
Cijena prema broju korisnika - poduzeće naplaćuje licence za svakog korisnika softvera drugog poduzeća. Kao primjer, *Microsoft* naplaćuje licencu *Windows* operacijskog sustava, za svakog korisnika. Također, ovaj način često daje i popust na količinu (eng. *Volume licencing*) kao što je slučaj kod *Microsofta (BIGCOMMERCE, 2024)*.

## 3.7. Ostali načini monetizacije

U ovom poglavlju navest će se načini monetizacije koji nisu puno zastupljeni, ali se također koriste dosta često.

### 3.7.1. Monetizacija pomoću partnerskog marketinga

Partnerski marketing (eng. *Affiliate marketing*) odnosi se na reklamiranje proizvoda drugih organizacija putem linkova, reklama i slika unutar aplikacije, kako bi se generirao prihod. Zapravo, generiranje prihoda radi se promoviranjem tuđih proizvoda, a prihod se dobiva kada korisnik odluči kupiti proizvod/uslugu. Partnerski marketing omogućuje odabir organizacija s kojima se želi stupiti u partnerstvo. Shodno tome, može se odlučiti reklamirati proizvode za koje se zna da će korisnici aplikacije biti zainteresirani. Izgled jedne *Banner* reklame koja je napravljena za partnerski marketing, prikazana je na slici 18.

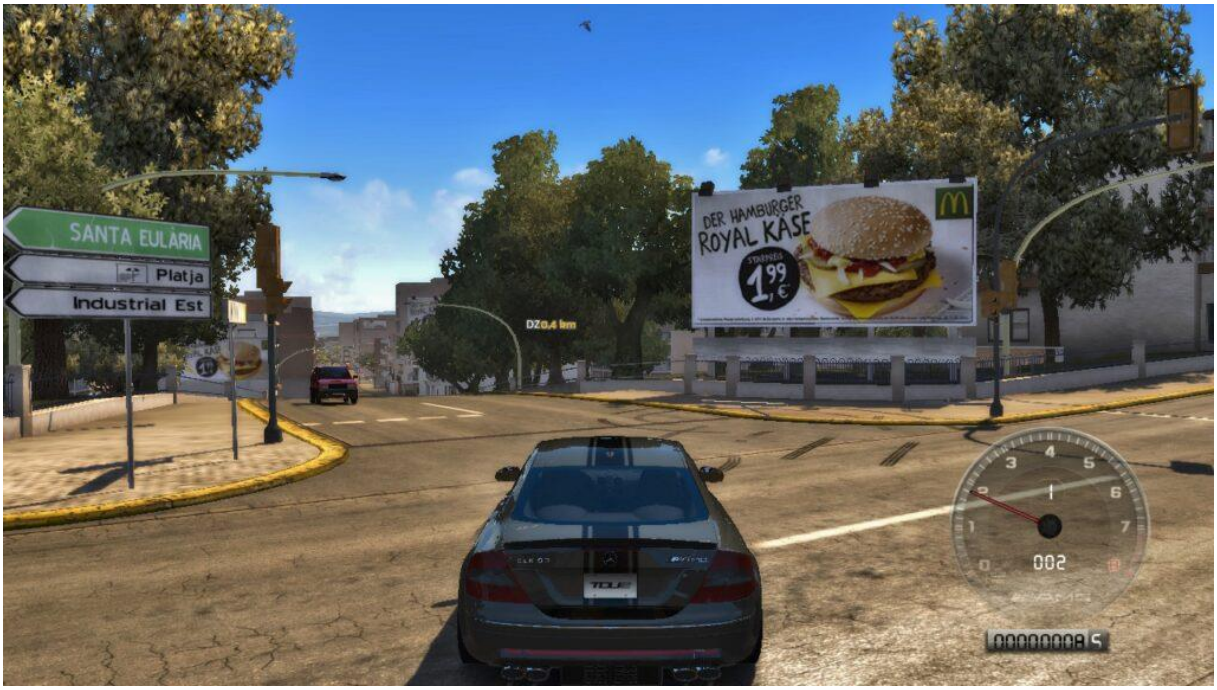



Slika 18: *Banner* reklama kod partnerskog marketinga (Hughes, 2024)

Bitno je napomenuti da ovakav način monetizacije u nekim slučajevima zahtjeva samostalnu izradu reklama. Organizacija čiji se proizvodi reklamiraju može dati samo smjernice, ali ne i reklamu. Drugim riječima, način na koji će se reklamirati proizvodi dan je na biranje izdavačima aplikacije (Beech, 2022).

### 3.7.2. Statične reklame unutar igara

Statične reklame unutar igara odnose se na reklamiranje proizvoda unutar igre. Bitno je naglasiti kako se ranije spomenute reklame (*Rewarded ads, Interstitial ads*) ne smatraju statičnim reklamama. Statične reklame podrazumijevaju smještanje proizvoda unutar „svijeta“ igre, te se ne mogu lagano promijeniti (Content & Insights Team, 2022). Navedeni način reklama nije toliko čest, posebice kod mobilnih igara, jer igra mora biti dovoljno popularna. Percipiranje brenda s lošim situacijama još je jedan od negativnih čimbenika. Kao primjer može se zamisliti da piće *Pepsi* pije glavni negativac u igri. *Pepsi* bi svakako htio izbjeći takvu reklamu. Osim na način da reklamirani proizvodi ili kompanije imaju direktnu interakciju s igračevim likom, logo brenda ili slika proizvoda može se staviti na reklamne ploče, ili na neki drugi način unutar igre.




Slika 19: Statična reklama u igri (Howards, 2012)

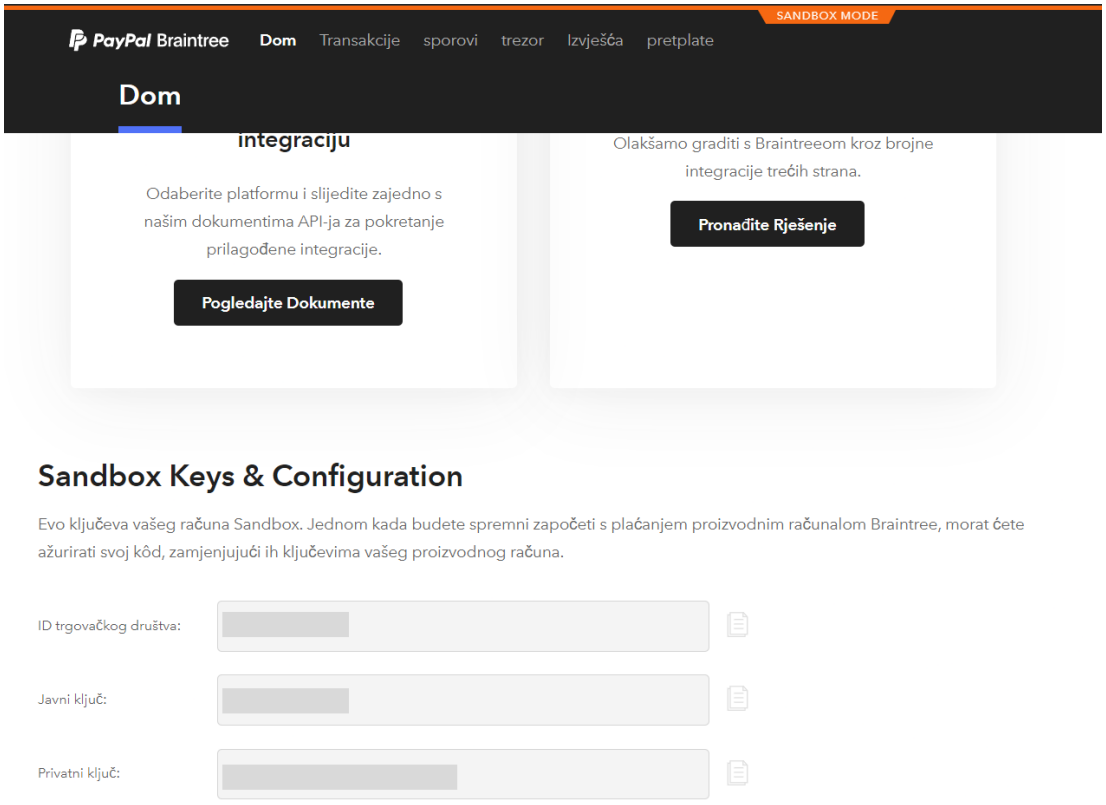
Na slici 19 vidljiva je reklama na reklamnoj ploči. Reklama prikazuje proizvod *McDonaldsa* sa cijenom i nazivom.

# 4. Praktični dio – *Braintree*

U praktičnom dijelu ovog rada odrađena je unutar aplikacijska kupovina. Sama implementacija napravljena je uz pomoć alata *Braintree*. *Braintree* je platforma koja omogućuje implementaciju unutar aplikacijskih kupovina. Kako bi se mogla razvijati i testirati funkcionalnost plaćanja, *Braintree* nudi „pješčanik“ (eng. *sandbox*). Ključevi koji se koriste u pješčaniku su drugi ključevi nego oni koji se koriste u produkciji.

## 4.1. Kreiranje računa, testnih ključeva i plaćanje

Kreiranje računa za *Braintree* „pješčanik“ može se odraditi na poveznici https://www.braintreepayments.com/hr/sandbox. Nakon kreiranja i verificiranja putem elektroničke pošte, prikazuje se zaslon s ključevima vidljiv na slici 20.




Slika 20: Zaslon nakon kreiranja računa (*Braintree*, 2024.)

Vidljiva su tri ključa:

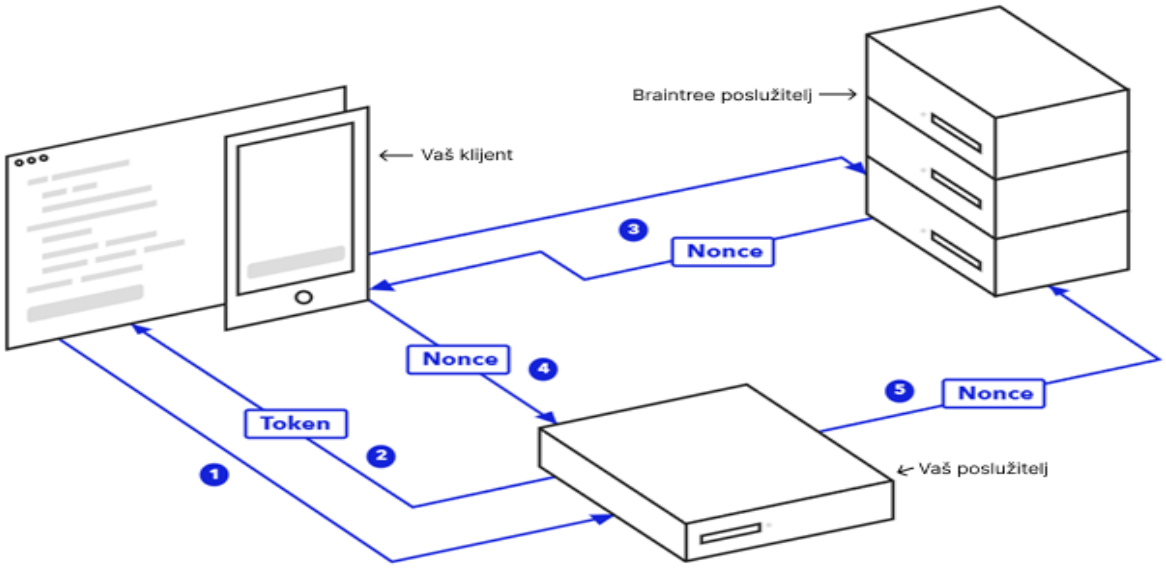
- *Merchant ID*

- *Public Key*
- *Private Key*

ID trgovca (eng. *Merchant ID)* je jedinstvena identifikacija profila, a javni (eng. *public)* i privatni (eng. *private*) ključevi su standardni ključevi koji omogućuju autentifikaciju i sigurnu komunikaciju između aplikacije i *Braintree* poslužitelja (enkripcija i dekripcija).

*Braintree* dijeli plaćanje na pet koraka (slika 21):

1. Zahtijevanje klijentskog tokena od servera i inicijalizacija *SDK-a* (PayPal Braintree, 2024a)
2. Server generira i šalje klijentski token klijentu (PayPal Braintree, 2024a)
3. Korisnik unosi podatke o plaćanju te *SDK* prenosi te informacije s *Braintreejem* i vraća *nonce* (PayPal Braintree, 2024a)
4. Klijentski dio šalje *nonce* serveru (PayPal Braintree, 2024a)
5. Server prima *nonce* i koristi ga za kreiranje transakcije (PayPal Braintree, 2024a)




Slika 21: Grafički prikaz načina rada *Braintree SDK-a* (PayPal Braintree, 2024a)

Na slici 21 vidljiv je grafički prikaz točaka koje su navedene ranije. Prvo se s klijentske strane šalje zahtjev za tokenom, a poslužiteljska strana aplikacije vraća token. Ovdje je važno

napomenuti da postoji način implementacije i bez klijentskog tokena koji će biti objašnjen u daljnjim poglavljima. Nakon toga, klijentska strana šalje zahtjev za *nonce* (eng. *Single-Use Payment Method*) te ga *Braintree* poslužitelj vraća klijentu. Ovdje je važno napomenuti kako se *nonce* dobiva uz pomoć *Drop-ina* u kojem korisnik unosi potrebne podatke. Nakon toga, klijentska strana šalje *nonce* poslužiteljskoj strani aplikacije, a ona šalje *nonce Braintree* serveru. Potom *Braintree* odrađuje transakciju (PayPal Braintree, 2024a).

## 4.2. Implementacija

Za postavljanje klijenta odabrana je verzija *Android v4*, budući da je klijentska aplikacija namijenjena mobilnim uređajima s *Android* operacijskim sustavom.

### 4.2.1. Postavljanje klijenta

Kao prvi korak, potrebno je dodati ovisnost na *Braintree drop-in* (kôd 20), a zatim dodati repozitorij s kojeg se mogu preuzeti potrebne ovisnosti u *build.gradle* na razini aplikacije (kôd 21).

```
implementation ("com.braintreepayments.api:drop-in:6.13.0")
```

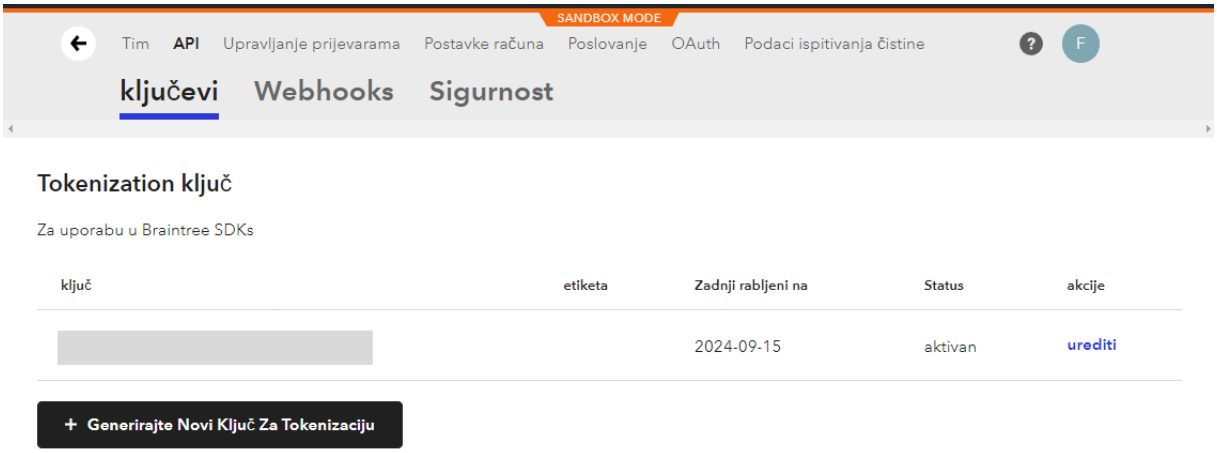
Kôd 20: Ovisnost za *DropIn* (Braintree, 2024)

```
allprojects { repositories {
        maven {
            url = uri("https://cardinalcommerceprod.jfrog.io/artifactory/android")
            credentials {
                username = "braintree_team_sdk"
                password =
"AKCp8jQcoDy2hxSWhDAUQKXLDPDx6NYRkqrgFLRc3qDrayg6rrCbJpsKKyMwaykVL8FWusJpp"
            }}}}
```

Kôd 21: Repozitorij (sarahkloop, 2021)

Dodavanje repozitorija nije definirano od strane *Braintreeja* na stranici koja objašnjava postavljanje. Također, putem interneta može se pronaći starija verzija koja neće raditi (guozchen-sh, 2021). Nadalje, potrebno je dobiti klijentski token koji se generira od strane poslužitelja ili, alternativno, koristiti *tokenization key* (Braintree, 2024). U praktičnom dijelu bit

će prikazana oba načina. *Tokenization key* generira se unutar *Braintree* konzole, klikom na *API* postavke prikazuje se zaslon na slici 22.




Slika 22: Postavke *API-ja* (*Braintree*, 2024)

Kako bi se generirao ključ, potrebno je kliknuti na gumb „Generirajte Novi Ključ Za Tokenizaciju“. Taj ključ može se koristiti umjesto klijentskog tokena. Klijentski token i *tokenization* ključ služe za autorizaciju i konfiguriranje detalja koji su potrebni za inicijalizaciju *SDK-a* na klijentskoj strani (Braintree, 2024). Za inicijalizaciju *DropInClineta* potrebno je inicijalizirati *DropInRequest* (kôd 22), a zatim se u *onCreate()* metodi poziva *setUpDropInClient()* (kôd 23).

```
private DropInRequest dropInRequest = new DropInRequest();
```

Kôd 22: Inicijalizacija *DropInRequesta* (Braintree, 2024)

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_donate);
        ButterKnife.bind(this);
        setUpDropInClient();
        setupListeners(); }
private void setUpDropInClient() {
        String authorization = "…";
        dropInClient = new DropInClient(this, authorization);
        dropInClient.setListener(this);}
```

Kôd 23: Inicijalizacija *DropInClienta* (Braintree, 2024)

Kod inicijalizacije *DropInClienta* na *Braintree* stranici, koja provodi kroz korake implementacije, koristi se drugi konstruktor. On kao parametre prima aktivnost, *DropInRequest* i *tokenization* ključ ili klijentski token. Takav pristup je zastario i više nije podržan (eng. *deprecated*). Iz tog razloga, koristi se novija verzija koja prima samo aktivnost i *tokenization* ključ ili klijentski token.

Kako bi se postavio iznos koji će korisnik platiti, dodani su gumbi koji prikazuju predefinirani izbor i gumb koji omogućuje korisniku da sam odredi iznos koji želi platiti. Nakon odabira iznosa, korisnik klikne na gumb „Plati“ te se prikazuje aktivnost *DropIn* koja vraća vrijednosti s obzirom na otvaranje i unos podataka o platnom sredstvu. Metoda *handlePaymentButtonClick()* odrađuje se prilikom klika na gumb „Plati“. Ona poziva metodu *isAmountValid()* koja poziva *getAmountFromEditText()* ako *amount* nije prazan. Nakon toga se pokreće *DropIn*. Kada je iznos prazan korisniku se prikazuje obavijest sa sadržajem „Molim unesite iznos!“ (kôd 24). Ako je iznos validan i uspješno se pokrene *DropIn* on se prikaže korisniku. Korisnik tada može izabrati način plaćanja (slika 23).


Slika 23: *DropIn*

Može se birati između nekoliko načina plaćanja, kao što je vidljivo na slici iznad. Ovaj zaslon prikazuje se uz pomoć kôda 24.

```
private void handlePaymentButtonClick(View view) {
        if (isAmountValid()) {
            dropInClient.launchDropIn(dropInRequest);
        } else {
            Toast.makeText(Obitelji3plusApplication.getContext(),
                           "Molim unesite iznos!", Toast.LENGTH_LONG).show();
        }
}
private boolean isAmountValid() {
        if (amount.isEmpty()) {
            getAmountFromEditText();
            return !amount.isEmpty();
        }
        return true;
}
private void getAmountFromEditText() {
    amount = String.valueOf(edTextAmount.getText());
}
```

Kôd 24: Pokretanje *DropIna* (Braintree, 2024)

Metoda *handlePaymentButtonClick()* provodi se kada korisnik klikne na gumb za plaćanje. Prvo se provjerava ako je korisnik odabrao ili upisao iznos. Ukoliko nije, prikazuje se obavijest na zaslonu koja ga obavještava da je potrebno odabrati iznos. Ako je iznos odabran i validan, pokreće se *DropInClient* pomoću metode *launchDropIn()* koja prima *DropInRequest* kao parametar. Zaslon koji se prikazuje nakon odabira kartice kao metode plaćanja, zahtijeva unos broja kartice i datuma isteka. Klikom na gumb *Add Card* šalje se zahtjev za *noncem* na *Braintree* poslužitelj. Nakon što se dobije odgovor, aktivnost se zatvara i tada se izvršava jedna od ovih metoda:

```
@Override
public void onDropInSuccess(@NonNull @NotNull DropInResult dropInResult) {
       String deviceData = dropInResult.getDeviceData();
       String nonce =
       Objects.requireNonNull(dropInResult.getPaymentMethodNonce()).getString();
       braintreeCheckout(nonce, deviceData);
}
@Override
public void onDropInFailure(@NonNull @NotNull Exception error) {
       Toast.makeText(Obitelji3plusApplication.getContext(),   "Drop   in   failure",
       Toast.LENGTH_LONG).show(); }
```

Kôd 25: Rezultat *DropIna* (Braintree, 2024)

Metoda *onDropInFailure()* izvršava se kada *Braintree* poslužitelj vrati grešku. To se može dogoditi kada, na primjer, nije postavljen *tokenization* ključ ili klijentski token. Također, metoda *onDropInFailure()* izvršava se kada korisnik prekine aktivnost. S druge strane, metoda *onDropInSuccess()* izvršava se nakon dobivanja *nonce* od *Braintree* poslužitelja, a nakon čega se mogu prikupiti podaci o uređaju korisnika. Nakon prikupljanja podataka šalje se *POST* zahtjev, uz pomoć *Retrofit2* biblioteke (kôd 26).

```
@FormUrlEncoded
@POST(URL.BT_CHECKOUT)
Call<Map<String, String>>
getBtreeCheckout(@Field(RequestValues.PAYMENT_METHOD_NONCE) String nonce,
@Field(RequestValues.DEVICE_DATA) String deviceData,
@Field(RequestValues.PAYMENT_AMOUNT) String amount);
```

Kôd 26: *POST* zahtjev (Braintree, 2024)

Kao što je već napomenuto, za postavljanje zahtjeva i njihovo slanje, koristi se biblioteka *Retrofit2*. Vidljivo je da se šalje *POST* zahtjev na *URL.BT_CHECKOUT*, čija vrijednost je vrijednost točke koja je definirana na poslužitelju. Unutar zahtjeva šalje se *nonce*, podaci o uređaju (eng. *device data*) i iznos (eng. *amount*). Odgovor se prosljeđuje metodi *generateReceipt()* klase *ReceiptGenerator* koja generira *pdf* datoteku i sprema ju u direktorij „Preuzimanja“ (eng. *downloads*). Korištena je biblioteka *Java IO*, standardne klase za formatiranje vremena, *Paint* klasa i *PdfDocument* klasa. Logika se može vidjeti u kôdu 27.

```
public class ReceiptGenerator {
    public static void generateReceipt(Context context, BtreePayment payment) {
        String type = getTypeText(payment.getType());
        String localDate = formatDateToLocalDate(payment.getDateTime());

        PdfDocument document = new PdfDocument();
        PdfDocument.PageInfo pageInfo = new PdfDocument.PageInfo.Builder(300, 600,
1).create();
        PdfDocument.Page page = document.startPage(pageInfo);

        page.getCanvas().drawText("Potvrda o plaćanju", 90, 50, new Paint());
        page.getCanvas().drawText("Iznos: " + payment.getAmount() + "€", 10, 110,
new Paint());
        page.getCanvas().drawText("Datum plaćanja: " + localDate, 10, 140, new
Paint());
```

```
        page.getCanvas().drawText("Razlog plaćanja: " + type , 10, 170, new
Paint());

        document.finishPage(page);

        String directoryPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS).getP
ath() + "/";
        File directory = new File(directoryPath);
        if (!directory.exists()) {
            directory.mkdirs();
        }

        String fileName = "receipt_" + payment.getPaymentId() + ".pdf";
        File file = new File(directoryPath, fileName);

        if (file.exists()) {
            String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss",
Locale.getDefault()).format(new Date());
            fileName = "receipt_" + payment.getPaymentId() + "_" + timeStamp +
".pdf";
            file = new File(directoryPath, fileName);
        }

        try {
            document.writeTo(new FileOutputStream(file));
            Toast.makeText(context, "Potvrda o plaćanju spremljena u: " +
file.getParent(), Toast.LENGTH_LONG).show();
        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(context, "Greška prilikom spremanja potvrde o plaćanju",
Toast.LENGTH_LONG).show();
        }
        document.close();
    }

    private static String formatDateToLocalDate(String dateTime) {
        SimpleDateFormat isoFormat = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSSX", Locale.getDefault());

        Date date = null;
        try {
            date = isoFormat.parse(dateTime);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        SimpleDateFormat localDateFormat = new SimpleDateFormat("dd.MM.yyyy.
HH:mm", Locale.getDefault());

        return localDateFormat.format(date);
    }

    private static String getTypeText(Integer type) {
        return type == 0 ? "Članarina" : "Donacija";
    }
}
```

Kôd 27: *ReceiptGenerator* klasa

Metoda *generateReceipt* prima parametre tipa *Context* i *BtreePayment*. *Context* omogućuje pristup stanju aplikacije. Kroz drugi parametar prosljeđuje se odgovor koji je dobiven od strane poslužitelja. Prvo se kreira prazni dokument tipa *pdf* uz pomoć *PdfDocument* klase, a nakon toga definiraju se dimenzije stranice. *PdfDocument.Page page = document.startPage(pageInfo)* postavlja novu stranicu unutar dokumenta sa prije definiranim dimenzijama. Samo „pisanje“ u dokument radi se metodama *page.getCanvas().drawText()* koja prima tekst koji će se ispisati, koordinate gdje će se tekst ispisati i *Paint* instanca. Pomoću nje može se stilizirati tekst. Metodom *finishPage()* završava se „pisanje“ u dokument. Nadalje se definira putanja do direktorija uz pomoć *getPath()* koja vraća putanju u obliku teksta (eng. *string*). Prvo grananje osigurava da direktorij postoji, to jest, ako ne postoji kreira se pomoću *mkdirs()* metode. Ako datoteka s istim imenom već postoji, tada se kao naziv postavlja identifikacijska oznaka transakcije i vrijeme transakcije. Spremanje dokumenta „umotano“ je u *try-catch* blok jer može rezultirati iznimkom. Pozivom *writeTo()* metode nad dokumentom on se sprema. Na kraju je potrebno zatvoriti dokument, a to omogućuje metoda *close()* koja se poziva nad dokumentom koji se želi zatvoriti. Metoda *formatDateToLocalDate()* prima datum i vrijeme, a vraća lokalizirani format, dok metoda *getTypeText()* vraća tip plaćanja. Drugim riječima, *getTypeText()* vraća vrijednost da li je korisnik platio članarinu ili je donirao. Generirani dokument vidljiv je na slici 24.


Slika 24: Generirani dokument

### 4.2.2. Postavljanje poslužiteljske aplikacije

Poslužiteljska strana pisana je u *PHP* programskom jeziku i koristi se programski okvir *Laravel*. Prije svega, potrebno je postaviti ovisnost (kôd 28), a zatim definirati točku (kôd 29).

```
{"require": { "braintree/braintree_php": "verzija" } }
```

Kôd 28: *Braintree* ovisnost *PHP* (Braintree, 2024)

```
Route::middleware('api')->post('/v3/btree_checkout', [BraintreeController::class,
'newCheckout']);
```

Kôd 29: *REST* točka za naplatu (Braintree, 2024)

Ako se ne koristi klijentski token, potrebna je samo jedna točka koja omogućuje komunikaciju između mobilne aplikacije i poslužitelja. Poslužitelj dalje komunicira s *Braintree* poslužiteljem kako bi se omogućila provedba plaćanja. U kôdu 29 je vidljivo da se koristi posrednički sloj (eng. *middleware*) naziva *api*. Ova ruta je za *POST* zahtjeve i nakon što se pošalje zahtjev, poziva se metoda *newCheckout()* u klasi *BraintreeController* koja je prikazana u kôdu 30.

```php
<?php
namespace App\Http\Controllers\Api\V3;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Braintree\Gateway;
use App\Http\Controllers\Api\V3\ApiController;
use Illuminate\Http\JsonResponse;
class BraintreeController extends ApiController
{
    private $gateway;
    private $isTokenValid = false;
    public function __construct()
    {
        parent::__construct();
        $this->gateway = new Gateway([
            'environment' => config('services.braintree.environment'),
            'merchantId' => config('services.braintree.merchant_id'),
            'publicKey' => config('services.braintree.public_key'),
            'privateKey' => config('services.braintree.private_key')
        ]);
    }
    public function newCheckout(Request $request)
    {
        $this->validateUserToken($request);
        if ($this->isTokenValid) {
            $nonceFromTheClient = $request->input('payment_method_nonce');
            $deviceDataFromTheClient = $request->input('device_data');
            $clientAmount = $request->input('amount');
            $isForMembership = $request->input('is_membership');
            $amount = floatval($clientAmount);
            $result = $this->gateway->transaction()->sale([
                'amount' => $amount,
                'paymentMethodNonce' => $nonceFromTheClient,
                'deviceData' => $deviceDataFromTheClient,
                'options' => [
                    'submitForSettlement' => true
                ]
            ]);
            if ($result->success) {
                $uplata = $this->recordPayment($result, $isForMembership);
                return response()->json($uplata, 200);
            } else {
                return response()->json(['message' => 'Payment failed!', 'error' =>
$result->message], 418);
            }
```

```
        } else {
            return response()->json(['message' => 'Unauthorized request.'], 403);
        }
    }
}
```

Kôd 30: *BraintreeController* (Braintree, 2024)

Za početak je nužno postaviti pristupnik (eng. *gateway*) preko kojeg se obavlja komunikacija s *Braintree* poslužiteljem. Okolina je postavljena na „pješčanik“, a *merchantId, publicKey* i *privateKey* postavljeni su na one ključeve koji se dobiju prilikom kreiranja *Braintree sandbox* korisničkog računa. Metoda *newCheckout()* čita vrijednosti *noncea, device_data* i *amounta* iz zahtjeva. Potom se uz pomoć metode *transaction()* podaci šalju *Braintree* poslužitelju kako bi se obavila transakcija. Nakon odgovora od strane *Braintree* poslužitelja, šalje se odgovor na mobilnu aplikaciju. U kôdu 31 prikazan je način na koji se bilježi uspjela transakcija te se ona šalje u odgovoru (kôd 30). Također, u kôdu 31 prikazana je i metoda *validateUserToken()* koja postavlja *isTokenValid* na temelju obrade *JWT-a* (eng. *JSON Web Token*).

```
private function validateUserToken(Request $request){
    $tokenValue = '';
    $response = new JsonResponse();
    if (!$this->parseAuthToken($request, $tokenValue, $response)) {
        $this->isTokenValid = false;
    } else if (!$this->checkTokenValidity($tokenValue, $response)) {
        $this->isTokenValid = false;
    } else {
        $this->isTokenValid = true;
    }
}
private function recordPayment($result, $isForMembership) : Uplata{
    $uplata = new Uplata;
    $uplata->braintree_id = $result->transaction->id;
    $uplata->iznos = $result->transaction->amount;
    $uplata->datum_vrijeme = Carbon::now();
    $uplata->tip = $isForMembership ? 0 : 1;
    $uplata->obitelji_id = 1;
    $uplata->save();
    return $uplata;
}
```

Kôd 31: Bilježenje transakcije i provjera *JWT* tokena

Metoda *validateUserToken()* koristi metode *parseAuthToken()* i *checkTokenValidity()* iz klase *ApiController* koja provjerava validnost *JWT-a*. Druga metoda je *recordPayment()* koja vraća tip *Uplata*. Ona generira novu *Uplatu* i vraća ju kako bi se mogla proslijediti u odgovoru za klijenta.

## 4.3. Korištenje klijentskog tokena

Kao što je već napomenuto, korisnički token generira *Braintree* poslužitelj i potrebno ga je generirati za svakog korisnika nakon isteka sesije. Kod poslužiteljske strane se za korištenje klijentskog tokena dodaje se još jedan *REST endpoint* kako bi klijent mogao zatražiti generiranje tokena od poslužitelja (kôd 32). Zahtjev se obrađuje u metodi *getClientToken()* (kôd 33). Klijentski token dobiva se od strane *Braintree* poslužitelja pozivanjem metoda *clientToken()* i *generate()*.

```
Route::middleware('api')->get('/v3/btree_token', [BraintreeController::class,
'getClientToken']);
```

Kôd 32: *REST* točka za klijentski token (PayPal Braintree, 2024c)

```
public function getClientToken(){
    $clientToken = $this->gateway->clientToken()->generate();
    return response()->json(['clientToken' => $clientToken]);
}
```

Kôd 33: Funkcija za generiranje klijentskog tokena (PayPal Braintree, 2024c)

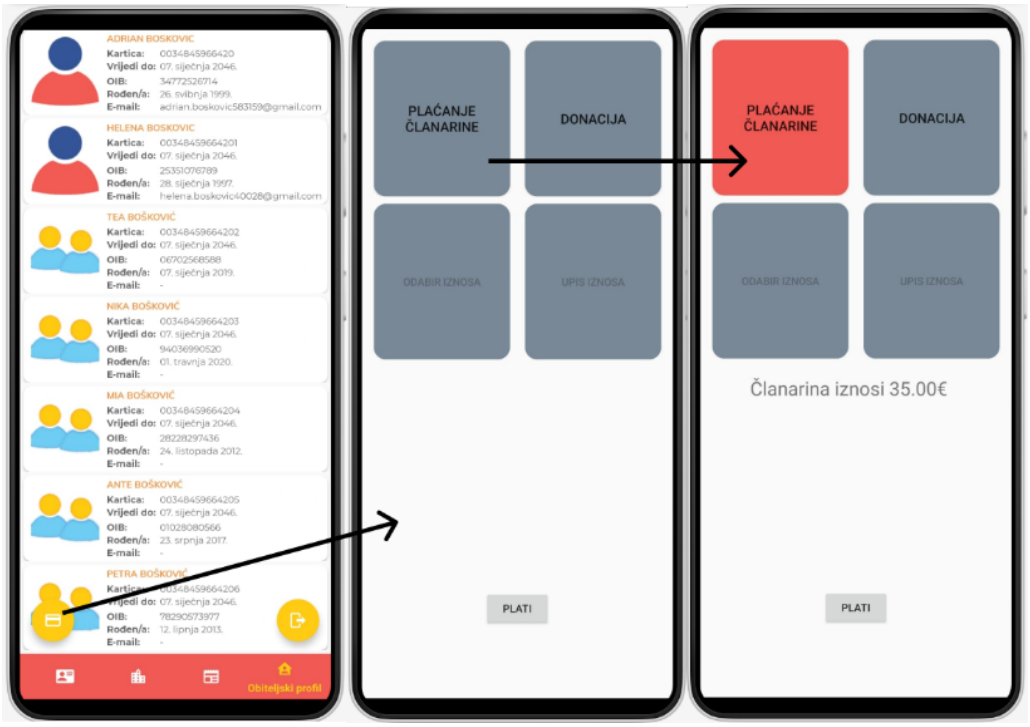
Kod klijenta, potrebno je poslati *GET* zahtjev na poslužitelja, kako bi se generirao token. Prilikom kreiranja *DropInClienta* umjesto *tokenization* ključa koristi se generirani klijentski ključ (kôd 34) (PayPal Braintree, 2024).

```
dropInClient = new DropInClient(this, clientTokenValue);
```

Kôd 34: Inicijalizacija *DropInClienta* sa klijentskim tokenom (PayPal Braintree, 2024c)

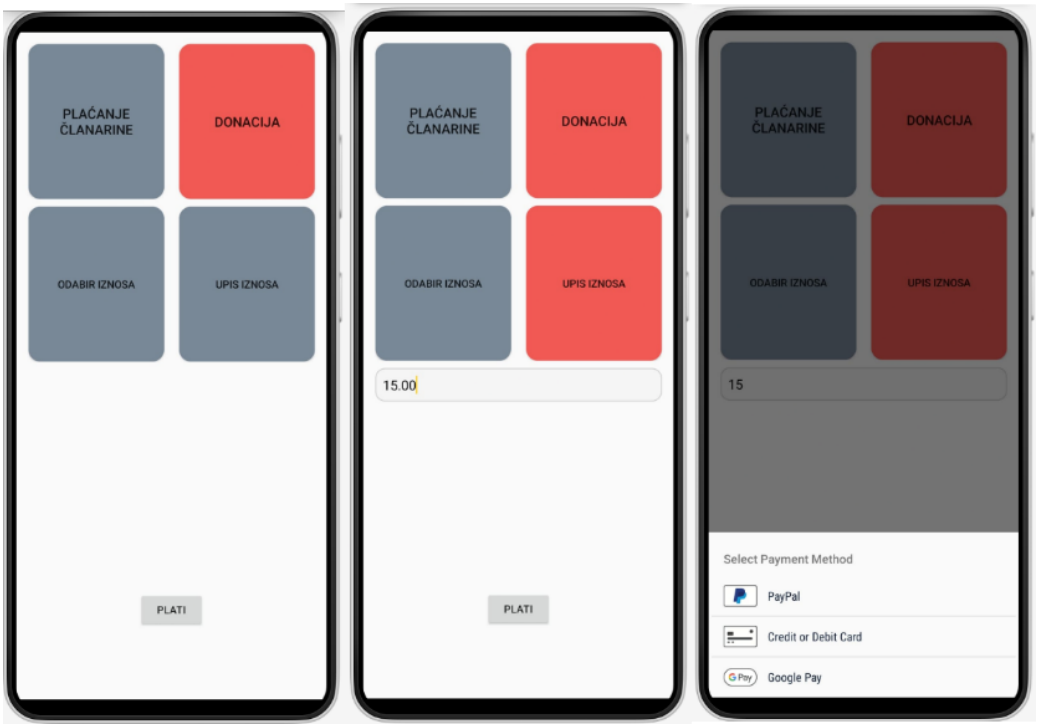
## 4.4. Slike zaslona

Ovo poglavlje prikazat će slike zaslona dijela aplikacije koji se tiče *Brainteeja* i prikazati će način prolaza kroz taj dio aplikacije. Slika 25 prikazuje navigaciju korisnika do aktivnosti sa implementiranim plaćanjem i odabir plaćanja članarine.



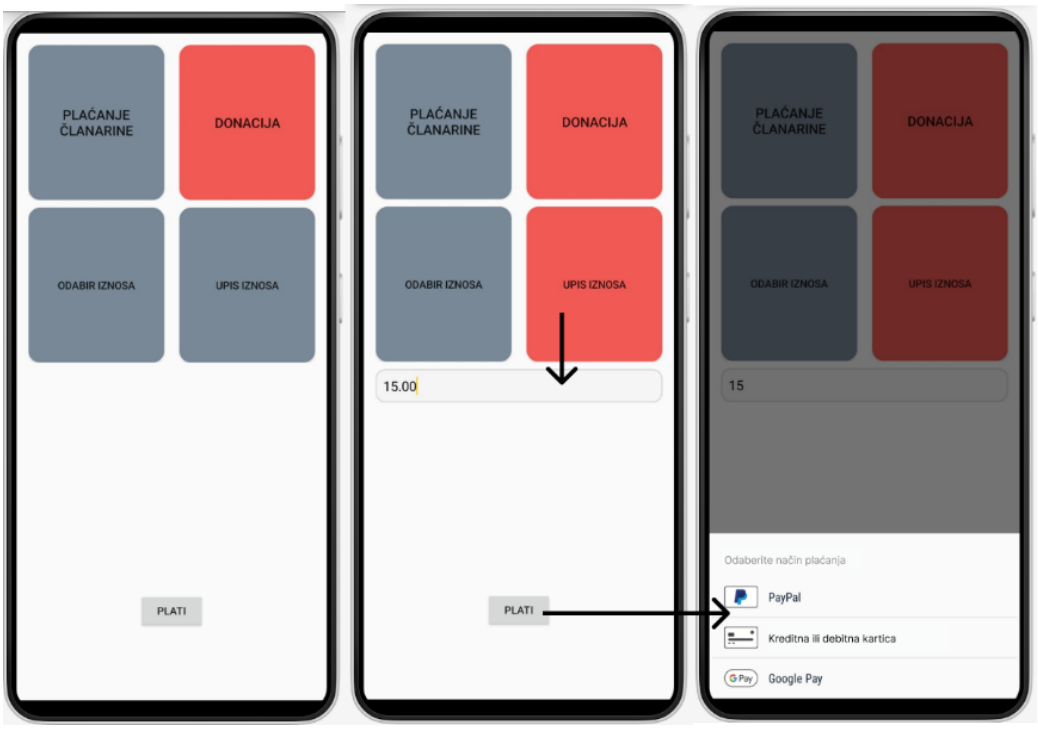

Slika 25: Plaćanje članarine

Klikom na gumb „plaćanje članarine“, korisniku se prikazuje cijena članarine u *TextView* elementu. Korisnik može pokrenuti plaćanje klikom na gumb „plati“ koji se nalazi na dnu ekrana. Klikom na gumb „donacija“, korisnik može odabrati ili upisati iznos (slika 26).




Slika 26: Opcija doniranja

Prilikom klika na gumb „Upis iznosa“ korisnik može samostalno upisati iznos koji želi donirati. Također, korisnik može odabrati iznos koji je već definiran klikom na gumb „odabir iznosa“. Nakon odabira, korisnik može pokrenuti plaćanje klikom na gumb „plati“ (slika 27).




Slika 27: Odabir definiranog iznosa

# 5. Zaključak

Monetizacija mobilnih aplikacija i igra, ali aplikacija općenito, je ključan aspekt koji doprinosi poslovanju izdavača aplikacija. Reklame su vrlo dobar način monetizacije mobilnih aplikacija i igara, jer omogućuju dobivanje prihoda od svakog korisnika koji koristi aplikaciju. *Google AdMob* nudi različite vrsta reklama, a implementacija nije previše zahtjevna. Unutar aplikacijske kupovine dobar su način za monetizaciju mobilnih aplikacija, a ponajviše mobilnih igara. *Google Billig* i *Google Console* veliki su i komplicirani sustavi za realizaciju unutar aplikacijskih kupovina, ali implementacija nije prezahtjevna. *Freemium* i pretplatnički modeli koriste se često zato što nude konstantan i predvidljiv prihod.

Praktični dio završnog rada odrađen je uz pomoć *Braintreeja* kako bi se implementiralo plaćanje unutar aplikacije. *Braintree* je vrlo jednostavan za implementirati i nudi podosta opcija plaćanja, kao što su *Google Pay*, plaćanje karticom, *PayPal* i *Samsung Pay*. No za razliku od *Google Consolea* i *Google Billinga*, *Braintree* nema način da se laganije definira što se kupuje, tako da je potrebno samostalno implementirati dio koji se bavi cijenom proizvoda. Radi toga, teže bi bilo realizirati prilagodbu cijene u odnosu na državu u kojoj se korisnik nalazi. Za kraj, *Braintree* je jako dobar alat za unutar aplikacijske kupovine i vrlo jednostavan za implementaciju.

# Popis literature


AdRoll. (2024). *Cost Per Mille (CPM): Meaning, Calculation and Applications | AdRoll*. Preuzeto 15. kolovoz 2024. od https://www.adroll.com/digital-advertising/cost-per-mille

adymob. (2024, srpanj 3). *Choosing the Best AdMob Ad Formats for Your App: A Complete Comparison! - Adymob*. Preuzeto 13. kolovoz 2024. od https://adymob.com/en/choosing-the-best-admob-ad-formats-for-your-app/

Akash Yadav. (2022, rujan 13). *Facebook Audience Network VS Google AdMob: A Comparison | by Akash Yadav | Medium*. Preuzeto 12. kolovoz 2024. od https://medium.com/@akashclp/facebook-audience-network-vs-google-admob-a-comparison-9aac6c4ecce4

Android Developers. (2024). *Kotlin coroutines on Android | Android Developers*. Preuzeto 28. kolovoz 2024. od https://developer.android.com/kotlin/coroutines

Apple Developer. (2024). *In-app purchase types - Reference - App Store Connect - Help - Apple Developer*. Preuzeto 15. kolovoz 2024. od https://developer.apple.com/help/app-store-connect/reference/in-app-purchase-types/

BIGCOMMERCE. (2024). *B2B Pricing Strategy: Different Models + Best Practices*. Preuzeto 30. kolovoz 2024. od https://www.bigcommerce.com/articles/b2b-ecommerce/b2b-pricing-strategy/

Braintree. (2024). *Set Up Your Client | Android - Braintree Developer Documentation*. Preuzeto 24. kolovoz 2024. od https://developer.paypal.com/braintree/docs/start/hello-client/android/v4

Business Model Toolbox. (2024). *Freemium - Business Model Toolbox*. Preuzeto 15. kolovoz 2024. od https://bmtoolbox.net/patterns/freemium/

Cambridge. (2024). *FREEMIUM | English meaning - Cambridge Dictionary*. Preuzeto 18. kolovoz 2024. od https://dictionary.cambridge.org/dictionary/english/freemium#google_vignette

Ceci, L. (2024). *Google Play: number of available apps 2009-2023*. Preuzeto 15. kolovoz 2024. od https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

Content & Insights Team. (2022, travanj 1). *What is in-game advertising in mobile? | Adjust*. Adjust. Preuzeto 30. srpanj 2024. od https://www.adjust.com/blog/what-is-in-game-advertising/

Content & Insights team. (2022, rujan 23). *A beginner's guide to app monetization | Adjust*. Preuzeto 28. srpanj 2024. od https://www.adjust.com/blog/how-to-monetize-your-app/

Genadinik, A. (2014). *Mobile app marketing and monetization*.

Google AdMob. (2024a). *Ad units, ad formats, & ad types - Google AdMob Help*. Preuzeto 15. kolovoz 2024. od https://support.google.com/admob/answer/6128738?hl=en

Google AdMob. (2024). *App Open Ads | Android | Google for Developers*. Preuzeto 15. kolovoz 2024. od https://developers.google.com/admob/android/app-open#java_3

Google AdMob. (2024b). *Banner ads | Android | Google for Developers*. Preuzeto 15. kolovoz 2024. od https://developers.google.com/admob/android/banner

Google AdMob. (2024c). *Get Started | Google for Developers*. Preuzeto 28. kolovoz 2024. od https://developers.google.com/admob/android/quick-start

Google AdMob. (2024d). *Payment thresholds - Google AdMob Help*. Preuzeto 15. kolovoz 2024. od https://support.google.com/admob/answer/2772208

Android Developers. (2024a, lipanj 10). *Integrate the Google Play Billing Library into your app | Google Play's billing system | Android Developers*. Preuzeto 30. srpanj 2024. od https://developer.android.com/google/play/billing/integrate

Google Developers. (2024b, kolovoz 15). *Rewarded ads | Android | Google for Developers*. Preuzeto 15. kolovoz 2024. od https://developers.google.com/admob/android/rewarded

guozchen-sh, sarahkloop. (2021, srpanj 8). *Could not resolve org.jfrog.cardinalcommerce.gradle:cardinalmobilesdk:2.2.3-2 due to 403 · Issue #414 · braintree/braintree_android · GitHub*. Preuzeto 3. kolovoz 2024. od https://github.com/braintree/braintree_android/issues/414

*Hrvatski jezični portal*. (2024). Preuzeto 15. kolovoz 2024. od https://hjp.znanje.hr/index.php?show=search_by_id&id=e1lvURA

Lindner, J. (2024, srpanj 17). *In App Purchase Statistics Statistics: Market Data Report 2024*. Preuzeto 13. kolovoz 2024. od https://gitnux.org/in-app-purchase-statistics/

Medina, H. (2024, travanj 25). *Need to know which ads pay more - Google AdMob Community*. Preuzeto 3. kolovoz 2024. od https://support.google.com/admob/thread/271033624/need-to-know-which-ads-pay-more?hl=en

Meta Business Help Center. (2024a). *About Meta Audience Network | Meta Business Help Center*. Preuzeto 15. kolovoz 2024. od https://web.facebook.com/business/help/788333711222886?_rdc=1&_rdr

Meta Business Help Center. (2024b). *About Meta Audience Network Payments | Meta Business Help Center*. Preuzeto 15. kolovoz 2024. od https://web.facebook.com/business/help/1664302973836952?_rdc=1&_rdr

*MONETIZATION | English meaning - Cambridge Dictionary*. (2024). Preuzeto 15. kolovoz 2024., od https://dictionary.cambridge.org/dictionary/english/monetization

Netflix. (2024). *Netflix*. Preuzeto 20. kolovoz 2024. od https://www.netflix.com/signup/planform

Osterwalder, A., & Pigneur, Y. (2010). *Business Model Generation*.

PayPal Braintree. (2024a). *Get Started - Braintree Developer Documentation*. Preuzeto 24. kolovoz 2024. od https://developer.paypal.com/braintree/docs/start/overview

PayPal Braintree. (2024b). *Overview - Braintree Developer Documentation*. Preuzeto 28. kolovoz 2024. od https://developer.paypal.com/braintree/docs/guides/authorization/overview

PayPal Braintree. (2024c). *Set Up Your Server | PHP - Braintree Developer Documentation*. Preuzeto 28. kolovoz 2024., od https://developer.paypal.com/braintree/docs/start/hello-server/php

Salter, T. (2022). Monetization. *Technological and Business Fundamentals for Mobile App Development*, 133–157. https://doi.org/10.1007/978-3-031-13855-3_6

sarahkloop. (2021, travanj 30). *Bintray Sunset - Cardinal Commerce Credentials · Issue #373 · braintree/braintree_android · GitHub*. GitHub. Preuzeto 3. kolovoz 2024. od https://github.com/braintree/braintree_android/issues/373

Seebacher, U. (2021). *B2B Marketing*.

Segal, T. (2024, lipanj 11). *Freemium: Definition, Examples, and Pros & Cons for Business*. Preuzeto 15. kolovoz 2024. od https://www.investopedia.com/terms/f/freemium.asp

Sharma, D. (2023, prosinac 22). *What is In-App Purchase (IAP)? Types of In App Purchases*. Preuzeto 11. kolovoz 2024. od https://adapty.io/blog/what-is-in-app-purchase/

Srinidhi, S. (2020, prosinac 19). *Free apps vs. Paid apps. Even though free apps are, well, free… | by Sunny Srinidhi | Medium*. Preuzeto 10. kolovoz 2024. od https://contactsunny.medium.com/free-apps-vs-paid-apps-b3e306092ae6

Tarver, E. (2024, travanj 19). *Subscription Business Model Defined, How It Works, Examples*. Investopedia. Preuzeto 2. kolovoz 2024. od https://www.investopedia.com/ask/answers/042715/how-do-subscription-business-models-work.asp

Tzuo, T., & Weisert, G. (2018). *Subscribed Why the Subscription Model Will Be Your Company's Future - and What to Do About It*.

Zaki, A. (2024, ožujak 15). *Subscription vs. Freemium: The Million Dollar Question for Your Mobile App | Medium*. Preuzeto 14. kolovoz 2024. od https://medium.com/@aadilzaki48/subscription-vs-freemium-for-your-mobile-app-a2e032600b96

Hughes, J. (2024, lipanj 10). *Create Banner Images for Your Affiliate Ads.* Preuzeto 14. rujan 2024. od https://easyaffiliate.com/blog/5-tools-to-help-you-create-banner-images-for-your-affiliate-ads/

Beech, I. (2022, ožujak 25.). *Affiliate marketing vs display advertising: which works best for my business*. Preuzeto 14. rujan 2024. od https://breezy.io/blog/affiliate-marketing-vs-display-advertising/#where-do-affiliate-marketing-and-display-advertising-overlap

Grimm, J. (bez dat.). *5 ways affiliate marketing benefits publishers [and boosts profits].* Preuzeto 14. rujan 2024. od https://breezy.io/blog/affiliate-marketing-vs-display-advertising/#where-do-affiliate-marketing-and-display-advertising-overlap

Howards, J. L. (2012, prosinac 5). *What Constitutes an Ad Impression for Dynamic In-Game Advertising.* Preuzeto 14 rujan 2024. od https://www.rapidfire.com/blog/what-constitutes-an-ad-impression-for-dynamic-in-game-advertising/

Ceci, L. (2024, ožujak 18). *Most popular installed ad network software development kits (SDKs) across Android apps worldwide as of March 2024.* Statista. Preuzeto 10. koloviz 2024. od https://www.statista.com/statistics/1035623/leading-mobile-app-ad-network-sdks-android/

Clement, J. (2021, rujan 28). *Rewards that mobile gamers in the United States are willing to watch ads for as of 2nd quarter 2020*. Statista. Preuzeto 10. kolovoz 2024. od https://www.statista.com/statistics/1266018/us-mobile-gamers-rewards-for-watching-ads/

AppsFlyer, (bez dat.). *New report on global in-app spending habits finds that Asian consumers spend 40% more in apps than the rest of the world*. Braze. Preuzeto 11. kolovoz 2024. od https://www.braze.com/resources/articles/in-app-purchase-stats

Ceci, L. (2023, ožujak 13). *Most popular installed commerce software development kits (SDKs) across Adnorid apps worldwide as of March 2023.* Statista. Preuzeto 10. kolovoz 2024. od https://www.statista.com/statistics/1036038/leading-mobile-app-commerce-sdks-android/

Ceci, L. (2024, ožujak 19). *Distribution of free and paid Android apps in the Google Play Store from June 2019 to March 2024.* Statista. Preuzeto 11. kolovoz 2024. od https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/

Enterprise Apps Today, (2023, prosinac 26). *Google Play Store Statistics 2024 – By Ratings, Demographics, Brand, Awareness, Device Traffic, Revenue, Paid App Price Distribution.* Preuzeto 13. kolovoz 2024. od https://www.enterpriseappstoday.com/stats/google-play-store-statistics.html

# Popis programskih kodova

# Popis slika

# Prilozi

1. Poveznica na repozitorij mobilne aplikacije:

   https://github.com/zstapic/obitelji3plus_mobile/tree/6-implementacija-pla%C4%87anja-unutar-aplikacije-uz-pomo%C4%87-braintreeja

2. Poveznica na repozitorij poslužiteljske aplikacije:

   https://github.com/zstapic/obitelji3plus_mobile/tree/poslu%C5%BEiteljska-strana-braintree-implementacije