

Sadržaj

Uvod	1
1. Pregled sličnih rješenja	2
1.1. 1Money	2
1.2. Money Manager	4
1.3. Snap & Split Bill	5
1.4. Zaključak analize	6
2. Specifikacija zahtjeva	7
2.1. Funkcionalni zahtjevi	7
2.2. Nefunkcionalni zahtjevi	9
3. Opis modela podataka	10
3.1. Konceptualni model	10
3.2. Firebase baza podataka	10
3.3. Opis baze podataka	11
3.4. Lokalno pohranjeni zapisi	12
4. Implementacija ključnih funkcionalnosti	14
4.1. Konfiguracija Firebase baze podataka	14
4.2. Prepoznavanje teksta sa slike	15
4.3. Prilagodba imena trgovina	16
4.4. Dohvaćanje trgovine iz Firebase baze podataka	17
5. Popis korištenih klasa i dodataka	18
5.1. Popis korištenih klasa	18
5.2. Popis korištenih dodataka (engl. dependencies)	21
6. Korisničke upute	23
6.1. Skeniranje zapisa računa	23
6.2. Ručni unos zapisa računa	24

6.3.	Dodavanje nove trgovine	25
6.4.	Pregledavanje zapisa računa	27
6.5.	Pregledavanje i uređivanje zapisa računa	28
6.5.1.	Uređivanje zapisa računa	28
6.5.2.	Dodavanje fotografije računa	30
6.6.	Dohvaćanje podataka u XLSX formatu	30
	Zaključak	31
	Literatura	32
	Sažetak	33
	Summary	34

Uvod

U današnje vrijeme praćenje financija postao je važan aspekt života modernoga čovjeka. Zbog razvoja online trgovina i kartičnog plaćanja, ljudi imaju sve manje doticaja s novcem u njegovom izvornom obliku, te je postalo vrlo izazovno pratiti sve troškove. Također, zbog sve ubrzanijeg načina života ljudi gube interes za komplicirane programe te očekuju od tehnologija da ponude što jednostavnija rješenja. Stoga su mobilne aplikacije u zadnjih 10 godina postale iznimno popularne. Zahvaljujući pokretljivosti mobilnih uređaja te njihovim mnogobrojnim senzorima (kamera, GPS, žiroskop, itd.) stvorile su se brojne mogućnosti za primjenu mobilnih aplikacija koje se nisu razmatrale u okviru programa za stolno računalo.

Iz navedenih razloga, javila se potreba za razvojem programskog proizvoda koji će omogućiti jednostavno i pregledno praćenje svih troškova na jednom mjestu. Idealno rješenje za to bila bi mobilna aplikacija koja bi koristila postupke računalnog vida. Skeniranje računa korištenjem tehnologije prepoznavanja teksta (engl. OCR) omogućuje korisnicima jednostavan unos podataka s računa u aplikaciju. Takva aplikacija osigurala bi korisnicima mobilnost, brzinu i fleksibilnost usluge, što je iznimno važno u kontekstu praćenja troškova u različitim svakodnevnim situacijama.

U okviru ovog završnog rada razvijena se mobilna aplikacija „*Scan Bill*“ za praćenje troškova. Aplikacija omogućava skeniranje računa i ukupne cijene te pohranu, praćenje i analizu potrošnje. U nastavku ovog rada bit će izložen pregled aplikacija sličnih funkcionalnosti, specifikacija funkcionalnih i nefunkcionalnih zahtjeva, opis modela podataka te definicija popisa korištenih klasa i paketa. Na kraju će biti navedene korisničke upute za aplikaciju, kroz koje će se demonstrirati funkcionalni zahtjevi aplikacije.

1. Pregled sličnih rješenja

Mobilnu aplikaciju u širem kontekstu možemo smatrati inovacijom. Kako bi inovacija bila uspješna i kako bi se mogla profitabilno komercijalizirati, važno je napraviti kvalitetnu analizu tržišta prije same implementacije. Tržište mobilnih aplikacija nalazi se u tzv. *online* trgovinama (engl. *store*) gdje se mogu pregledavati i instalirati aplikacije za pojedini mobilni uređaj.

U ovom poglavlju bit će iznesen pregled dvije najpopularnije aplikacije na Google Play-u kojima je cilj evidencija troškova te najpopularnija aplikacija koja ima funkcionalnost skeniranja računa. Usporedit će se funkcionalnosti aplikacija, navesti prednosti i mane aplikacije, napraviti analiza dizajna tih aplikacija te analizirati korisničko iskustvo (engl. *user experience*) prilikom korištenja tih aplikacija. Sve analizirane aplikacije mogu se besplatno preuzeti s Google Play trgovine. Dolje navedena tablica (Tablica 1.1) daje kratki pregled tih aplikacija te najvažnije informacije o njima.

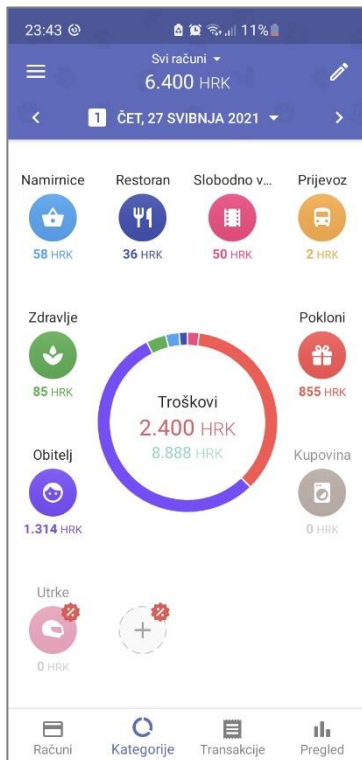
Tablica 1.1 kratki pregled analiziranih aplikacija

Ime aplikacije	Proizvođač	Ocjena	Broj preuzimanja
1Money	PixelRush	4.7 / 5	+ 1 000 000
Money Manager	Realbyte Inc.	4.7 / 5	+ 10 000 000
Snap & Split Bill	Standy Software	4.2 / 5	+ 10 000

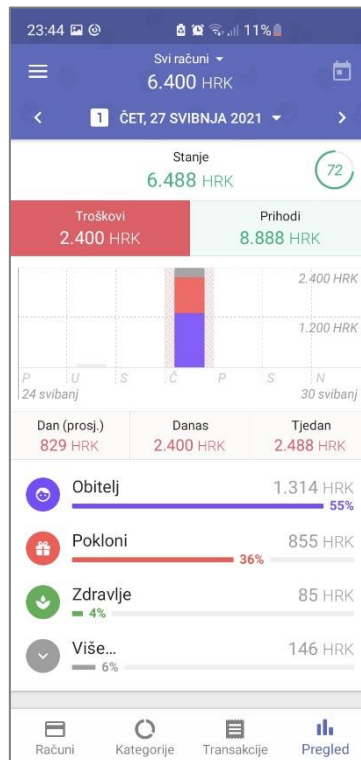
1.1. 1Money

1Money je aplikacija za praćenje troškova. U opisu aplikacije navodi se da aplikacija omogućava sljedeće: jednostavni način praćenja troškova, brzi izračun proračuna i funkcionalnosti osobnog financijskog menadžera [1]. 1Money je u početku besplatan, ali unutar aplikacije postoje stavke i funkcionalnosti koje su onemogućene dok se ne kupi cijela verzija aplikacije. Pri korištenju aplikacije vidljivo je da je glavni način raspodjele troškova po kategorijama (npr. namirnice, restorani, slobodno vrijeme, prijevoz i slično) te da se mogu pratiti i prihodi (gotovinske i kartične transakcije). U donjem izborniku nalazi se nekoliko mogućih pogleda na troškove i prihode: u izborniku *Kategorije* dostupan je pregled po kategorijama (Slika 1.1), u izborniku *Transakcije* navedene su sve transakcije za izabrani

period, a u izborniku *Pregled* dostupan je stupčasti dijagram koji pokazuje potrošnju u odnosu na određeni vremenski period te ostale relevantne podatke (Slika 1.2). Jedini način unosa iznosa potrošnje je korištenjem tipkovnice te se pri tome može izabrati kategorija u koju spada trošak (Slika 1.3).



Slika 1.1 Kategorije



Slika 1.2 Pregled

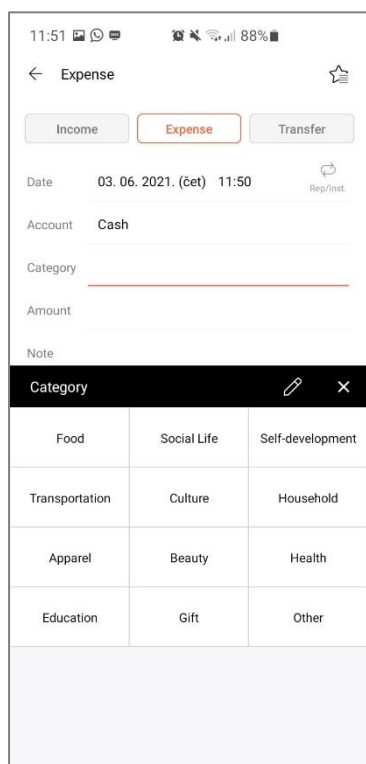
Slika 1.3 unos troška

Neke od najznačajnijih prednosti aplikacije su njezina preglednost (ikone za svaku kategoriju su vrlo jasne i jednoznačne), implementirana podrška za većinu svjetskih valuta te dostupnost korištenja na nekoliko jezika. Glavni nedostatak aplikacije je potreba za unosom preko tipkovnice. U nekim slučajevima to može biti komplicirano i kod korisnika izazivati nepotreban napor. Današnje moderne aplikacije teže što jednostavnijem korištenju i automatizaciji što većeg broja radnji te bi bilo prikladno ponuditi korisniku mogućnost skeniranja računa.

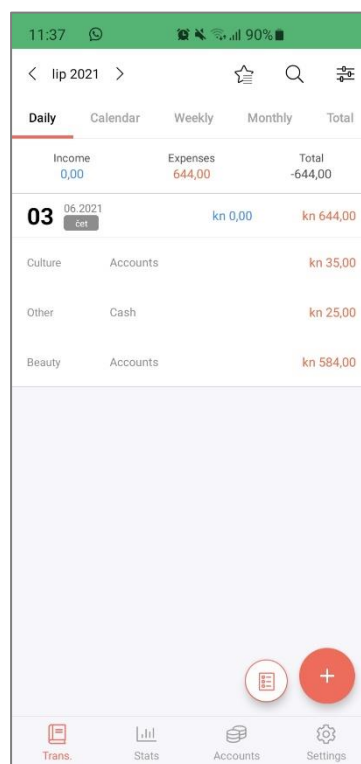
Korisničko sučelje je vrlo dobro te nudi pregršt različitih mogućnosti i izbora. Jedini nedostatak u dizajnu je nepregledan i nejasan graf u izborniku *Pregled*.

1.2. Money Manager

Funkcionalnost aplikacije *Money Manager* vrlo je slična funkcionalnosti aplikacije *IMoney*. Njezini autori navode da je zauzela prvo mjesto na ljestvici aplikacija za financijsko planiranje, praćenje troškova i financijsko upravljanje [2]. *Money Manager* omogućava unos troškova putem tipkovnice koji se mogu filtrirati po načinu plaćanja (kartica ili gotovina) i po kategoriji troškova. Kategorije su vrlo općenite te ne pružaju dovoljno raznoliku kategorizaciju troškova (Slika 1.4). Aplikacija je vrlo jednostavna i nudi poprilično uzak spektar mogućnosti. S jedne strane, nekim korisnicima to može predstavljati nedostatak jer nema dovoljno funkcionalnosti da zadovolji njihove potrebe. S druge strane, korisnicima koji od aplikacije za praćenje troškova očekuju da bude jednostavna i bez previše nepotrebnih dodataka koje neće niti koristiti, upravo to predstavlja najveću prednost ove aplikacije. Jedna od mana aplikacije je njezina neintuitivnost. Elementi grafičkog korisničkog sučelja nisu optimalno raspodijeljeni po ekranu, a boje i veličine fontova loše su usklađeni te mogu korisnika koji prvi puta upotrebljava ovu aplikaciju ostaviti poprilično zbunjenim (Slika 1.5). Međutim, to je donekle razumljivo s obzirom na činjenicu da je aplikacija dostupna u trgovini od 2013. godine, otkada su tehnologije i standardi razvoja aplikacija doživjeli ogroman napredak.



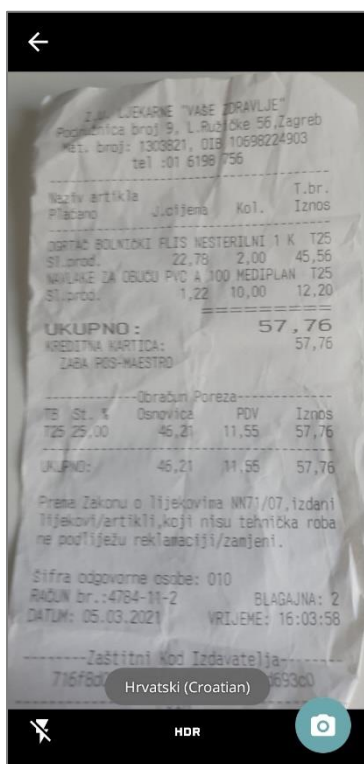
Slika 1.4 izbor kategorija



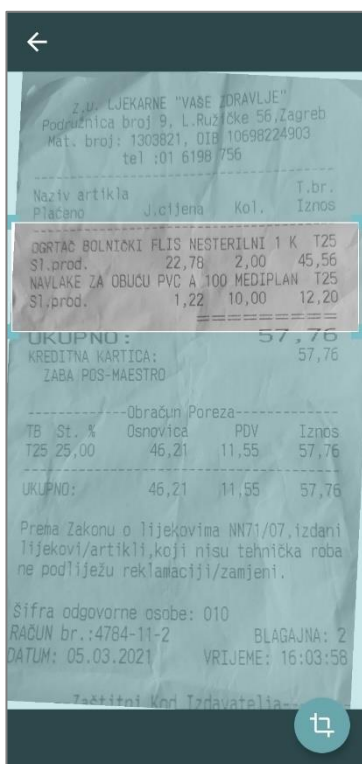
Slika 1.5 pregled troškova

1.3. Snap & Split Bill

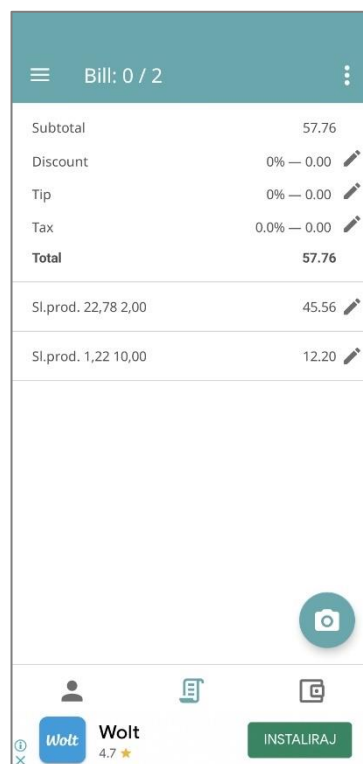
Snap & Split Bill aplikacija namijenjena je skeniranju računa i dijeljenju troškova među unesenim korisnicima [3]. Iako ova aplikacija po svojoj funkcionalnosti nije posve slična funkcionalnosti aplikacije kojom se bavi ovaj završni rad, postoje neke sličnosti, a glavna od njih je mogućnost skeniranja računa. Skeniranje računa u aplikaciji *Snap & Split Bill* provodi se u nekoliko faza. Prvi korak je fotografiranje računa koji želimo podijeliti (Slika 1.6) Nakon toga, korisniku se nudi opcija izrezivanja dijela slike na kojem se nalaze podaci o troškovima (Slika 1.7). Na posljetku, obavlja se analiza slike i korisnik nakon nekoliko sekundi dobiva informacije o troškovima (Slika 1.8) Međutim, konačan rezultat skeniranja dostupan je tek u posljednjem koraku opisanog ireverzibilnog postupka te ako korisnik nije zadovoljan rezultatom mora ponovo proći kroz sve korake kako bi se skeniranje pravilno obavilo.



Slika 1.6 slikanje računa



Slika 1.7 izrezivanje računa



Slika 1.8 prikaz troškova

Upravo taj problem stvara komplikacije pri korištenju i nije usklađen s načelima jednostavnosti aplikacija. Cijeli postupak skeniranja slike je poprilično redundantan s obzirom na činjenicu da u današnje vrijeme postoje napredne tehnologije prepoznavanja teksta (engl. *OCR*) koje omogućuju prepoznavanje teksta u stvarnom vremenu (engl. *Real-*

time). Njihovom upotrebom pojednostavio bi se proces skeniranja, a korisnik bi odmah dobio informaciju o tome što je skener prepoznao.

1.4. Zaključak analize

Navedene aplikacije samo su neka od postojećih programskih rješenja koja implementiraju funkcionalnost praćenja troškova. Analizom je utvrđeno da ima mnogo sličnih varijacija na temu, ali svima njima je zajednička karakteristika da je unos moguć isključivo preko mobilne tipkovnice. To u suštini ne predstavlja problem pri korištenju aplikacije, ali zasigurno nije optimalno rješenje. Zbog ubrzanog razvoja tehnologija kao što su umjetna inteligencija (engl. *artificial intelligence*) i strojno učenje (engl. *machine learning*), korisnici se sve više oslanjaju na te tehnologije te samim time očekuje da aplikacije ponude takva rješenja gdje je to moguće. Stoga se uočava potreba da se korisnicima ponudi aplikacija koja će imati mogućnost skeniranja računa. Iako postoje aplikacija koje nude funkcionalnosti skeniranja teksta, odnosno računa, taj postupak skeniranja vrlo je kompliciran i neusklađen s mogućnostima skeniranja u stvarnom vremenu.

Na temelju gore navedenih teza jasno se naslućuje potreba tržišta za inovacijom u obliku aplikacije koja će korisnicima koji žele pratiti svoje troškove na pametnom uređaju omogućiti jednostavnije praćenje troškova. Aplikacija bi omogućila korisnicima skeniranje računa i prepoznavanje dijelova računa u stvarnom vremenu kroz moderan i jednostavan dizajn.

2. Specifikacija zahtjeva

Temeljna zadaća programskog rješenja je ostvariti zahtjeve koji su proizašli iz analize sličnih programskih rješenja. Cilj takvog pristupa je ponuditi korisnicima rješenje koje će biti naprednije i inovativnije od postojećih proizvoda. Mobilna aplikacija pri tome mora zadovoljiti temeljne zahtjeve modernih mobilnih programa kao što su responzivnost, intuitivnost, jasnoća te moderan i minimalistički dizajn. Nadalje, aplikacija mora biti multiplatformska, to jest sve komponente moraju ispravno raditi na najpopularnijim mobilnim operacijskim sustavima: Androidu i iOS-u. Aplikacija za rad treba biti povezana na internet.

2.1. Funkcionalni zahtjevi

1. Prepoznavanje teksta

Mobilna aplikacija treba omogućiti prepoznavanje teksta automatski pomoću kamere. Prepoznavanje teksta treba se obavljati bez potrebe da se slika s kamere sprema na uređaj i tako usporava cijeli proces.

2. Prilagođavanje skeniranog teksta

Potrebno je istražiti i primijeniti mogućnosti prilagodbe teksta za brže pretraživanje baze podataka. Cilj prilagodbe je pronaći funkciju koja će domenu svih mogućih imena trgovina jednoznačno preslikati na konačni skup mogućih naziva trgovina koji će biti pohranjeni u bazi podataka.

3. Pronalaženje trgovine u bazi podataka koja je dostupna na Internetu

Prilagođene nazive skeniranih tekstova treba moći pretražiti u bazi podataka. Baza podataka mora biti dostupna na internetu i mora indeksirati nazive trgovina.

4. Detektiranje ukupne cijene

Nakon što je prepoznata trgovina, korisniku se omogućava odabir cijene. Dinamički se prikazuju sve skenirane brojčane vrijednosti koje su zapisane u jednom od formata prikaza cijene.

5. Spremanje zapisa računa u internu memoriju

Nakon skeniranja trgovine i odabira cijene, korisniku se mora ponuditi mogućnost spremanja zapisa u memoriju. Zapis dodatno sadrži podatak o vremenu unosa i kategoriji trgovine koja se dohvaća iz baze podataka. Prilikom spremanja moraju se ažurirati svi pogledi na podatke.

6. Ručni unos zapisa računa

Potrebno je omogućiti korisniku unos zapisa koristeći mobilnu tipkovnicu. Prilikom unosa trgovine korisnik bira trgovinu koja se nalazi u bazi podataka. Korisnik ima mogućnost dodavanja nove trgovine prilikom unosa. Prilikom spremanja zapisa, potrebno je provjeriti jesu li unesene ispravne vrijednosti za ime trgovine i cijenu. Ime trgovine je validno ako postoji u bazi podataka.

7. Dodavanje nove trgovine u bazu podataka

Korisniku treba omogućiti dodavanje nove trgovine prilikom skeniranja računa ili ručnog unosa. Prilikom dodavanja nove trgovine, korisnik unosi njezin naziv, tekst imena trgovine koji se nalazi na računu te odabire kategoriju za trgovinu. Informacije o dodanoj trgovini spremaju se u bazu podataka te samim time ta trgovina postaje dostupna ostalim korisnicima za skeniranje ili ručni unos.

8. Prikaz spremljenih zapisa i statistike za određen vremenski period

Korisnik treba moći dohvatiti sve zapise za određeni vremenski period. Vremenski period može biti dan, mjesec ili godina. Dohvaćene podatke potrebno je prikazati u listi i kreirati stupčasti i kružni dijagram. U kružnom dijagramu treba omogućiti grupiranje po trgovini i kategoriji.

9. Uređivanje spremljenog zapisa

Korisniku je potrebno omogućiti uređivanje odabranog zapisa iz liste zapisa. Opcija uređivanja zapisa treba ponuditi mogućnost promjene naziva trgovine, cijene i datuma tog zapisa troška. Prilikom izmjene, vrijede ista ograničenja koja vrijede prilikom dodavanja novog zapisa. Također, potrebno je omogućiti brisanje zapisa.

10. Dodavanje slike postojećem zapisu

Prilikom uređivanja zapisa, korisnik ima mogućnost dodavanje slike računa. Slika računa se sprema zajedno sa zapisom u internu memoriju korisnika.

11. Dohvaćanje podataka u XLSX formatu

Aplikacija treba omogućiti korisniku preuzimanje i dijeljenje zapisa o računima u XLSX formatu.

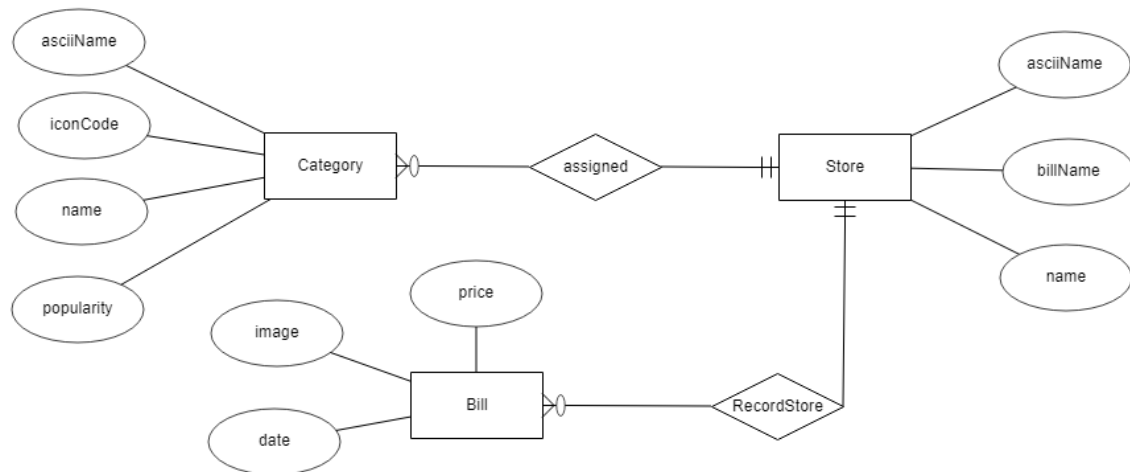
2.2. Nefunkcionalni zahtjevi

- Aplikacija je namijenjena globalnom tržištu, stoga korisničko sučelje treba biti pisano na engleskom jeziku.
- Aplikacija ne smije biti ograničena na lokalne postavke (valuta, izgled računa, diakritički znakovi i slično).
- Aplikacija se razvija u razvojnom okviru (engl. framework) Flutter [4] kako bi bila multiplatformska.
- Pretraživanje baze podataka mora biti složenosti $O(\log N)$ kako bi pretraživanje baze podataka bilo brzo i efikasno.
- Logo aplikacije mora svojim dizajnom biti usklađen s izgledom aplikacije.
- Skeniranje računa mora trajati ispod 100 ms kako bi korisnikovo iskustvo bilo fluidno tijekom korištenja aplikacije.
- Sve vrijednosti s decimalnom točkom moraju biti zaokružene na jednu decimalu radi preglednijeg prikaza na dijagramima.

3. Opis modela podataka

3.1. Konceptualni model

U nastavku je prikazan konceptualni model baze podataka (Dijagram 3.1).



Dijagram 3.1 konceptualni model podataka

3.2. Firebase baza podataka

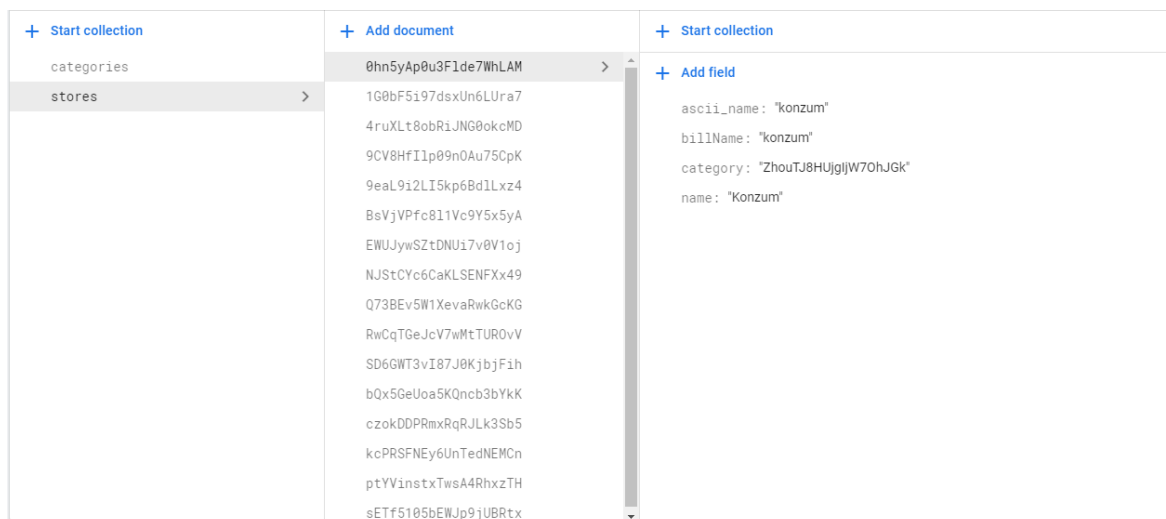
Mobilna aplikacija „Scan The Bill“ sastoji se od programskog modula koji se izvodi na klijentskom uređaju i baze podataka. Baza podataka organizirana je kao *Backend-as-a-Service (BaaS)*. *Backend-as-a-Service* je popularno rješenje računarstva u oblaku (engl. *cloud-computing*) u kojem je pozadinska aplikacija (engl. *backend*) dostupna u obliku usluge (engl. *services*) na internetu. Takvu uslugu uglavnom pružaju za to specijalizirane kompanije koje svakom korisniku dodjele dio svog sklopovlja te nude aplikacijsko programsko sučelje (engl. *API*) za komunikaciju s tim sklopovljem. Prednosti korištenja takvog pristupa su brojne, a jedna od njih je ta da kompanije koje se bave razvojem aplikacija ili web usluga ne moraju voditi brigu o upravljanju bazom podataka i odražavanju pozadinske infrastrukture. U ovo završnom radu za pohranu podataka koristi se *Backend-as-a-Service Firebase* baza podataka.

Firebase je programska razvojna platforma pokrenuta 2011. godine koju je 2014. godine kupio Google i nastavio razvijati za svoje potrebe [5]. Prvobitno je bila zamišljena kao baza podataka u stvarnom vremenu (engl. *real-time database*), a danas nudi 18 različitih usluga.

Neke od najznačajnijih prednosti *Firebase* baze podataka je to što je besplatna, nudi vrlo temeljitu dokumentaciju te ima pristupačno korisničko sučelje. U ovom završnom radu koriste se sljedeće *Firebase* usluge: *Cloud Firestore* za bazu podataka i *ML Kit*, o kojem će biti više riječi u poglavljima koja slijede.

Cloud Firestore

Cloud Firestore je *NoSQL* baza podataka koja omogućava jednostavno spremanje, sinkroniziranje i postavljanje upita nad bazom podataka [6]. Podržana je za Android, iOS i Web aplikacije. Baza podataka sastavljena je od kolekcija (engl. *collections*) koje sadrže dokumente (engl. *documents*) koji predstavljaju jedan entitet u bazi podataka (Slika 3.1). Dokument može sadržavati polja (engl. *fields*) i kolekcije. Polje po svom tipu može biti *string*, *number array* i slično. *Cloud Firestore* automatski indeksira sva unesena polja kako bi pretraživanje bilo učinkovito i brzo.



Slika 3.1 primjer *Cloud Firestore* baze podataka

3.3. Opis baze podataka

Kao što je već navedeno, za bazu podataka koristi se *Firebase* usluga *Cloud Firestore*. Baza podataka sastavljena je od dvije kolekcije *categories* i *stores*. Kolekcija *categories* (Tablica 3.1) sadrži popis dokumenata koji predstavljaju kategorije trgovina. Kategorije trgovina ciljano su izvedene kao dio baze podataka kako bi dodavanje novih kategorija trgovina bilo jednostavno i bez potrebe za mijenjanjem kôda. Kolekcija *stores* (Tablica 3.2) namijenjena je spremanju trgovina. Trgovine dodaju korisnici aplikacije i jednom dodana trgovina vidljiva je svim ostalim korisnicima prilikom skeniranja ili ručnog unosa.

Tablica 3.1 Kolekcija *categories*

Ime polja	Tip polja	Opis polja
asciiName	string	Prilagođeno ime kategorije, koristi se kako se prilikom pretraživanja kategorija ne bi radila razlika između velikih i malih slova (engl. <i>case insensitive</i>)
iconCode	number	Kôd ikone kategorije
name	string	Ime kategorije
popularity	number	Broj trgovina kojima je dodijeljena ta kategorija

Tablica 3.2 Kolekcija *stores*

Ime polja	Tip polja	Opis polja
asciiName	string	Prilagođeno ime trgovine, koristi se kako se prilikom pretraživanja trgovina ne bi radila razlika između velikih i malih slova (engl. <i>case insensitive</i>)
billName	string	Prilagođeno ime trgovine koje se nalazi na računu, nastalo kao rezultat postupka prilagodbe teksta (postupkom Prilagođavanje skeniranog teksta)
category	string	ID kategorije kojoj pripada trgovina
name	string	Ime trgovine

3.4. Lokalno pohranjeni zapisi

Korisnik unesene račune sprema u internu memoriju uređaja. Prilikom prvog spremanja, kreira se datoteka u CSV formatu koja se pohranjuje u direktorij koji je svakoj aplikaciji dodijeljen od strane operacijskog sustava. Opis podataka u datoteci dan je u Tablica 3.3.

Tablica 3.3 vrijednosti stupaca u lokalno spremljenoj CSV datoteci

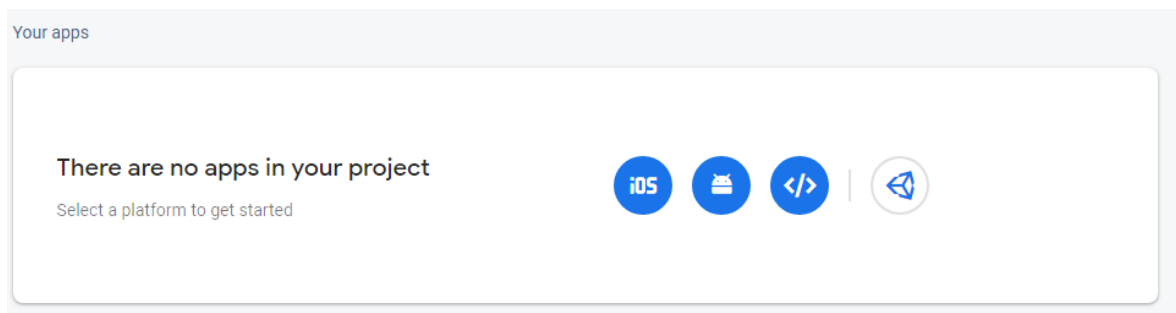
Naziv vrijednosti stupca	Opis vrijednosti stupca
date	Datum zapisa, inicijalni datum je datum dodavanja računa u datoteku
store	Ime trgovine
category	Kategorija kojoj trgovina pripada
categoryIcon	Kôd ikone kategorije
price	Ukupna cijena računa
image	Slika računa spremljena u binarnom zapisu, postavlja se kada korisnik prilikom uređivanja zapisa doda sliku

4. Implementacija ključnih funkcionalnosti

U ovom poglavlju slijedi objašnjenje ključnih funkcionalnosti aplikacija. Bit će navedeni dodaci korišteni prilikom implementacije zadanih funkcionalnosti te prikazani najbitniji dijelovi kôda.

4.1. Konfiguracija Firebase baze podataka

Za korištenje Firebase baze podataka potrebno je imati Google račun. Pritiskom na gumb konzola (engl. *console*) prijavljeni korisnik ima pristup svojim projektima te može kreirati novi projekt. Sam postupak povezivanja Firebase s programskim rješenjem vrlo je jednostavan. Za povezivanje Firebase i Flutter projekta najprije je potrebno kreirati novi projekt u Firebase konzoli te u projekt dodati aplikaciju (Slika 4.1)



Slika 4.1 Dio Firebase Console za dodavanje aplikacije u projekt

Postupak dodavanja aplikacije implementiran je u 3 koraka. U prvom koraku unose se podaci o aplikaciji, kao što su ime paketa (engl. *package name*) i ime aplikacije. Nakon toga, u sljedećem koraku Firebase na temelju unesenih podataka generira jedinstvenu datoteku koju je potrebno dodati u posebnu mapu unutar projekta. Zadnji korak je unos dodataka koji su potrebni za rad Firebase u aplikaciji. U uputama je vrlo jasno navedeno koje dodatke je potrebno dodati i gdje ih je potrebno smjestiti. Završetkom trećeg koraka aplikacija je povezana s Firebase-om te je moguće komunicirati s bazom podataka pomoću metoda iz dodataka `cloud_firestore`. Način kreiranja upita vrlo je dobro dokumentiran u službenoj dokumentaciji [7].

4.2. Prepoznavanje teksta sa slike

Za prepoznavanje teksta sa slike koristi se dodatak (engl. *dependency*) *flutter_camera_ml_vision* [8]. *Flutter_camera_ml_vision* dekorira funkcionalnosti *Firestore ML-a* (*Firestore Machine Learning*) [9] na način da nudi *Flutter widget* [10] koji prikazuje tok podataka slika s kamere (engl. *camera stream*), primjenjuje detekciju pomoću *Firestore ML-a* te na posljetku vraća objekt tipa `VisionText` u metodi `onResult` (Kôd 4.1). U metodi iz objekta tipa `VisionText` dohvaća se prepoznati tekst, dijeli po operatoru za novi red te se poziva asinkrona metoda za obradu teksta `onScanResult`. Zbog činjenice da izvođenje metode `onScanResult` može trajati relativno dugo (zbog pristupa bazi podataka) odbacuju se svi zahtjevi za obradom u metodi `onResult` dok ne završi poziv prošle metode `onScanResult`. Na taj način korisnik nema vizualni dojam da se u pozadini odvija proces koji može trajati relativno dugo.

```
onResult: (VisionText text) {  
    if (!mounted) {  
        return;  
    }  
    if (sl.dispose != true &&  
        sl.previousFinish != false &&  
        sl.isVisible != false) {  
        sl.previousFinish = false;  
        sl.onScanResult(text).then((value) {  
            if (value == true) {  
                setState(() {});  
            }  
            sl.previousFinish = true;  
        });  
    }  
},
```

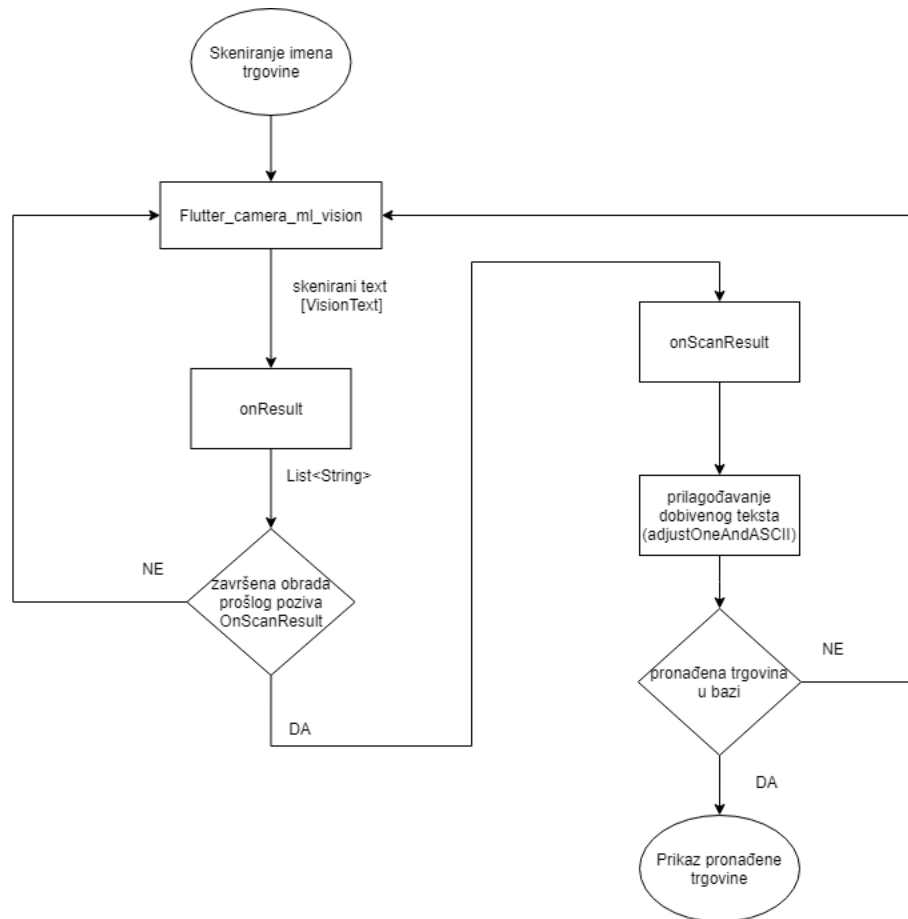
Kôd 4.1 metoda `onResult`

4.3. Prilagodba imena trgovina

Prije implementacije funkcionalnosti aplikacije provedeno je testiranje prepoznavanja teksta na velikom broju računa. Zbog nekonzistentnosti dizajna računa i fonta slova uočeno je da prepoznavanje teksta ima problema s nekim kombinacijama slova. Na primjer, ako račun sadrži kombinaciju slova „ri“, u većini slučajeva ta kombinacija će biti protumačena u slovo „n“. Također, mnoge trgovine u imenima sadrže dijakritičke znakove i brojeve (mala razlika između 0 i O). Takve pogreške dovele bi do nemogućnosti prepoznavanja računa koje sadrže takve kombinacije slova. Stoga, sve linije teksta najprije prolaze kroz metodu koja prilagođava dobiveni tekst po zadanim pravilima koja su proizašla iz rezultata testiranja. Ta metoda sve slične kombinacije svodi na jednu od kombinacija kako bi prepoznavanje bilo jednoznačno. Ova prilagodba se primjenjuje i prilikom dodavanja nove trgovine u bazu te se prilagođena vrijednost zapisuje u polje *billName*. Za zamjenu dijakritičkih znakova metoda se oslanja na dodatak *diacritic* [11] koji u metodi `removeDiacritics` zamjenjuje sve dijakritičke znakove s osnovnim znakovima, a njezin kôd prikazan je u odjeljku Kôd 4.2. Cijeli postupak skeniranja i prepoznavanja trgovine dan je u Dijagram 4.1.

```
String adjustOneAndASCII(String s, Map<String, String> rules) {  
    s = s.trim();  
    s = s.toLowerCase();  
    s = removeDiacritics(s);  
    rules.forEach((key, value) {  
        s = s.replaceAll(key, value);  
    });  
    return s;  
}
```

Kôd 4.2 implementacija funkcije prilagodbe imena trgovina



Dijagram 4.1 dijagram toka analize imena trgovine

4.4. Dohvaćanje trgovine iz Firebase baze podataka

Nakon što se dobiveni tekstovi prilagode, šalje se zahtjev *Firebase* bazi podataka da vrati sve trgovine čije se polje *billName* podudara s prilagođenim tekstovima. *Firebase* omogućava kreiranje upita za provjeru podudaranja vrijednosti polja s nekim elementom predane liste uz ograničenje da predana lista smije sadržavati maksimalno 10 elemenata. *Firebase* upit se postavlja nad referencom koja pokazuje na jednu od početnih kolekcija.

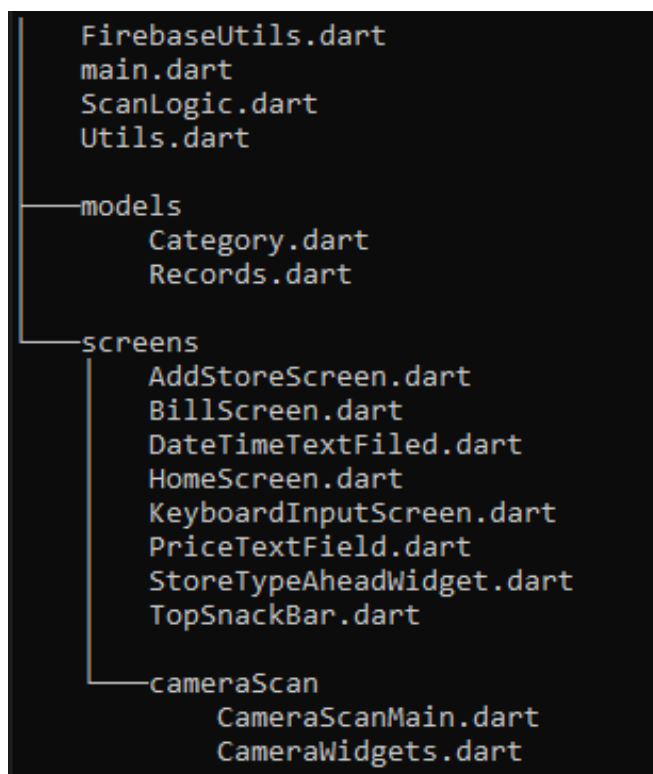
```
CollectionReference categoryCollectionReference =
    FirebaseFirestore.instance.collection('categories');
```

Nad referencom je moguće obavljati određene akcije, primjerice dodavati *where* upite i postavljati ograničenja na broj vraćenih dokumenata. Sam rezultat upita je *Future* objekt kojem se može pristupiti u funkciji u kojoj se upit izvršava ili delegirati *FutureBuilder widget* da animira korisničko sučelje dok podaci nisu dostupni.

5. Popis korištenih klasa i dodataka

Pri razvoju velikih programa i aplikacija koji se definiraju kroz više klasa potrebno je od samog početka voditi računa o organizaciji projekta po direktorijima i datotekama.

Aplikacija je podijeljena na način da su u korijenskom direktoriju smještene sve programske datoteke koje obavljaju programsku logiku. U direktoriju *screens* nalaze se dijelovi programa koji definiraju korisničko sučelje, dok se u direktoriju *models* nalaze klase koje predstavljaju pojedine modele u bazi ili internoj memoriji (Slika 5.1).



Slika 5.1 Organizacija aplikacije po direktorijima

5.1. Popis korištenih klasa

- **main.dart**

Pri pokretanju aplikacije poziva se metoda `main` koja se nalazi unutar *main.dart* datoteke. Inicijaliziraju se svi potrebni resursi i definira se glavna tema programa (boja, font i sl.). Program se dalje usmjerava na *HomeScreen.dart*.

- **Utils.dart**

U datoteci *Utils.dart* nalaze se pomoćne metode koje obavljaju različite postupke transformacije, prilagodbe i filtracije.

- **FirestoreUtils.dart**

Konfiguracija komunikacije s bazom podataka *Firestore* definirana je u datoteci *FirestoreUtils.dart*. U toj datoteci nalaze se asinkrone metode koje izvršavaju upite nad bazom ili mijenjaju sadržaj baze podataka.

- **ScanLogic.dart**

Proces skeniranja može se nalaziti u jednom od tri stanja: skeniranje imena trgovine, skeniranje ukupne cijene ili stanje čekanja da korisnik potvrdi unos. Ponašanje programa kroz ta stanja definirano je u datoteci *ScanLogic.dart*. Nakon dobivanja rezultata skeniranja, poziva se metoda `onScanResult` koja se nalazi u ovoj datoteci.

- **Category.dart**

Category.dart predstavlja model podataka koji je u bazi podataka definiran kolekcijom *categories*.

- **Records.dart**

Model podataka zapisa računa koji se sprema u lokalnu memoriju predstavljen je razredom `Records` u datoteci *Records.dart*, u kojoj se nalaze i metode za učitavanje podataka iz CSV datoteke te njihovo uređivanje i brisanje u datoteci.

- **HomeScreen.dart**

Dizajn početnog zaslona aplikacije ostvaren je u datoteci *HomeScreen.dart*. Na početnom zaslonu nalazi se selektor vremenskog perioda (Slika 5.2), lista svih zapisa za taj period te okrugli i stupčasti dijagram. U donjem lijevom kutu nalazi se lebdeći gumb za dodavanje novih zapisa računa.



Slika 5.2 selektor vremenskog perioda

- **CameraScanMain.dart**

U datoteci *CameraScanMain.dart* ostvaren je dizajn zaslona skeniranja. Zaslون je definiran u dva sloja. Na donjem sloju prikazana je kamera, dok se na gornjem sloju nalazi dio za odabir trgovine i cijene.

- **CameraWidgets.dart**

Dizajn stanja u kojem se može način proces skeniranja ostvaren je u datoteci *CameraWidgets.dart*.

- **KeyboardInputScreen.dart**

Unos računa pomoću virtualne tipkovnice dizajniran je u datoteci *KeyboardInputScreen.dart*.

- **AddStoreScreen.dart**

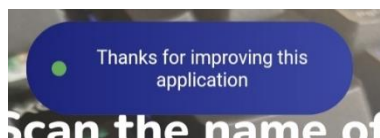
AddStoreScreen.dart definira plutajući prozor za dodavanje novih trgovina.

- **BillScreen.dart**

Zaslون na kojem korisnik ima mogućnost uređivanja zapisa računa realiziran je u datoteci *BillScreen.dart*

- **TopSnackBar.dart**

U datoteci *TopSnackBar.dart* ostvaren je plutajući prozor u kojem se korisniku prikazuju poruke (Slika 5.3).



Slika 5.3 lebdeći prozor za prikaz poruke

- **DateTimeTextField.dart**

DateTimeTextField.dart definira polja za unos datuma i vremena pri uređivanju zapisa. U datoteci su definirane i posebne tipkovnice za unos datuma i vremena.

- **PriceTextField.dart**

Datoteka *PriceTextField.dart* ostvaruje dekorator oko *widget*-a polja za unos. Definira izgled i pravila za validaciju polja za unos cijene.

- **StoreTypeAheadWidget.dart**

Datoteka *StoreTypeAheadWidget.dart* ostvaruje dekorator oko *widget*-a polja za unos imena trgovine. Omogućuje prikaz postojećih imena trgovine na temelju korisničkog unosa.


5.2. Popis korištenih dodataka (engl. dependencies)



- cupertino_icons: ^1.0.2 (https://pub.dev/packages/cupertino_icons) – ikone korištene u aplikaciji
- flutter_camera_ml_vision: ^2.2.4 (https://pub.dev/packages/flutter_camera_ml_vision) – skeniranje teksta iz toka podataka
- cloud_firestore: ^1.0.1 (https://pub.dev/packages/cloud_firestore) – upiti prema Firebase bazi podataka
- firebase_core: ^1.0.1 (https://pub.dev/packages/firebase_core) – korijenski dodatak za rad s Firestore bazom podataka
- keyboard_avoider: ^0.1.2 (https://pub.dev/packages/keyboard_avoider) – dodatak za dizajn ekrana prilikom pojavljivanja tipkovnice
- diacritic: ^0.1.3 (<https://pub.dev/packages/diacritic>) – metode za pretvorbu dijakritičkih znakova
- flutter_typeahead: ^3.1.2 (https://pub.dev/packages/flutter_typeahead) – Widget polje za unos koji nudi padajući izbornik opcija
- flushbar: ^1.10.4 (<https://pub.dev/packages/flushbar>) – lebdeći statusni widget koji se koristi za prikaz poruka korisniku
- bubble_tab_indicator: ^0.1.5" (https://pub.dev/packages/bubble_tab_indicator) – Widget za izbor vremenskog okvira (dan, mjesec, godina)
- date_util: ^0.1.4 (https://pub.dev/packages/date_util) – metode za rad s datumima
- date_utils: 0.2.0 (https://pub.dev/packages/date_utils) – metode za pretvorbu mjeseca u tekstualni oblik (npr. 7. mjesec → srpanj)
- path_provider: any (https://pub.dev/packages/path_provider) – spremanje datoteke na određenu lokaciju
- csv: ^5.0.0 (<https://pub.dev/packages/csv>) – rad s CSV formatom datoteka
- pie_chart: 4.0.1 (https://pub.dev/packages/pie_chart) – Widget za kružni dijagram

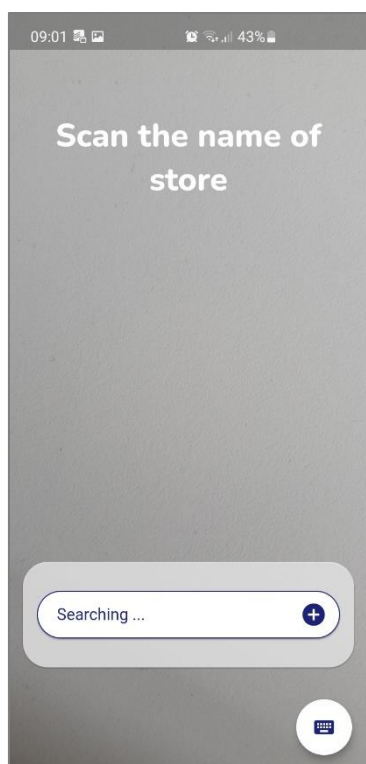
- collection: ^1.15.0 (<https://pub.dev/packages/collection>) – metoda za grupiranje kolekcije
- charts_flutter: ^0.10.0 (https://pub.dev/packages/charts_flutter) – Widget za stupčasti dijagram
- excel: ^1.1.5 (<https://pub.dev/packages/excel>) – metode za rad s Excel datotekom
- share: ^2.0.1 (<https://pub.dev/packages/share>) – metoda za dijeljenje datoteka
- image_picker: ^0.7.5 (https://pub.dev/packages/image_picker) – metode za baratanje sa slikom
- full_screen_image: ^1.0.2 (https://pub.dev/packages/full_screen_image) – Widget za prikaz slike na cijelom ekranu
- flutter_cupertino_date_picker: ^1.0.26+2
(https://pub.dev/packages/flutter_cupertino_date_picker) – prikaz datumske i vremenske tipkovnice
- rename: ^1.3.1 (<https://pub.dev/packages/rename>) – promjena imena projekta

6. Korisničke upute

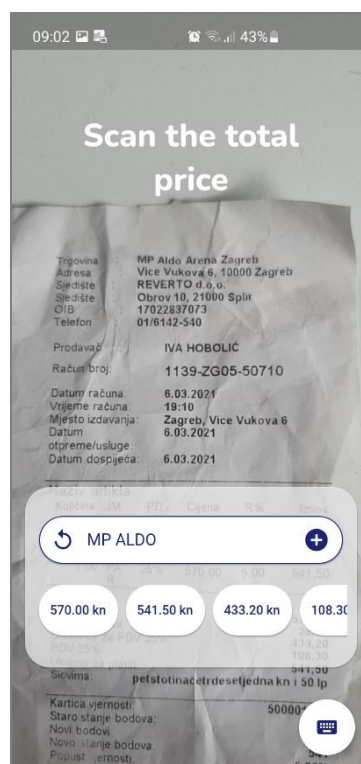
6.1. Skeniranje zapisa računa

Pritiskom gumba  u desnom donjem kutu na početnom zaslonu pokreće se skeniranje računa. Na zaslonu je ispisana poruka „Scan the name of the store“ te aplikacija analizira skenirani tekst i pretražuje imena trgovina u bazi (Slika 6.1). Kada je trgovina uspješno pronađena, pokreće se skeniranje cijena, prikazuje se poruka „Scan the total price“ te se cijene dinamički prikazuju na zaslonu (Slika 6.2). Nakon što korisnik izabere točnu ukupnu cijenu, aplikacija prestaje skeniranje te prikazuje gumb „Submit“ (Slika 6.3). Uklanjanjem selekcije, aplikacija nastavlja skenirati cijene. Pritiskom na gumb „Submit“, zapis računa dodaje se u memoriju, korisnik se vraća na početni zaslon i ispisuje se poruka „New record added“ (Slika 6.4).

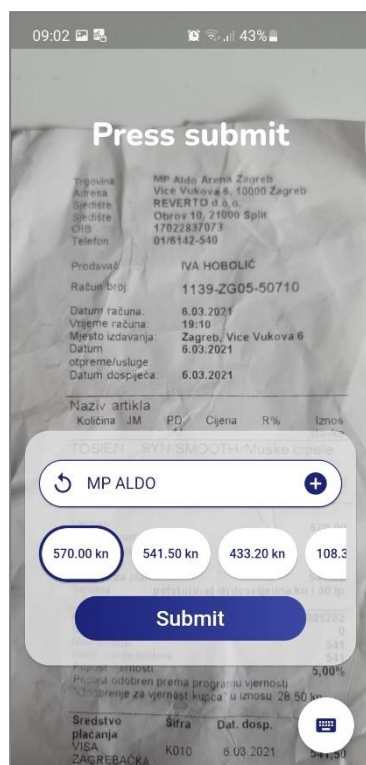
Ako trgovina ne postoji u bazi podataka, korisnik ju može direktno dodati iz zaslona skeniranja odabirom opcije . Pritiskom na gumb  korisnik poništava odabir skenirane trgovine te se aplikacija vraća u stanje pretraživanja trgovina. Pri tome, poništena trgovina ne prikazuje se ponovo tijekom tog pretraživanja.



Slika 6.1 skeniranje trgovine



Slika 6.2 skeniranje cijene




Slika 6.3 odabir cijene

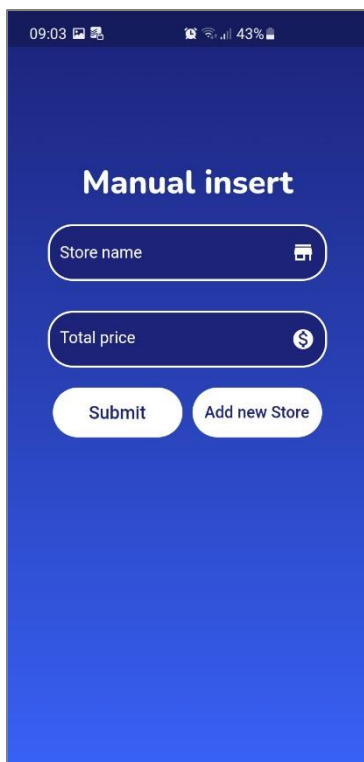


Slika 6.4 poruka o uspješno dodanom novom zapisu

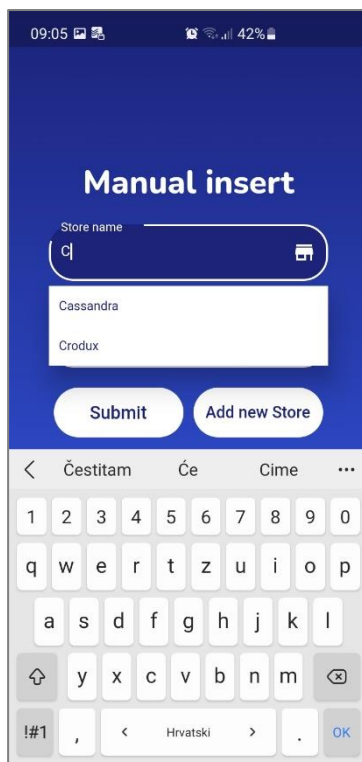
6.2. Ručni unos zapisa računa

Ukoliko nije moguće skenirati račun, korisnik ima mogućnost ručnog unosa zapisa računa.

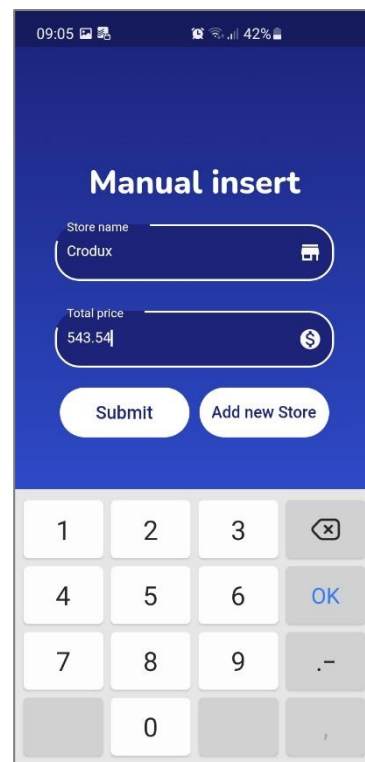
Pritiskom na gumb , koji se nalazi u desnom donjem kutu na zaslonu za skeniranje, prikazuje se stranica za ručni unos imena trgovine i cijene (Slika 6.5). Korisnik najprije unosi ime trgovine, prilikom čega se u padajućem izborniku prikazuju imena trgovina koje počinju s korisničkim unosom (Slika 6.6). Korisnik mora izabrati jednu od ponuđenih trgovina, odnosno nije moguće dodati trgovinu koja se ne nalazi u bazi podataka. Nakon odabira trgovine, korisnik unosi ukupnu cijenu (Slika 6.7). Kada su sva polja ispravno unesena, pritiskom na gumb „Submit“ zapis se dodaje u memoriju te se izvodi ista procedura dodavanja zapisa kao u slučaju dodavanja skeniranog računa.



Slika 6.5 ručni unos zapisa računa



Slika 6.6 padajući izbornik imena trgovina



Slika 6.7 unos cijene

6.3. Dodavanje nove trgovine

Aplikacija je zamišljena tako da i korisnici doprinose njezinom razvoju na način da dodaju trgovine i tako omogućuju drugima automatizirano skeniranje istih. Trgovinu je moguće dodati na više mjesta kako bi se potaknulo korisnike da dodaju nove trgovine. Kada se odabere dodavanje nove trgovine, na ekranu se prikazuje plutajući zaslon koji korisnika vodi kroz postupak dodavanja nove trgovine (Slika 6.8). Najprije se unosi ime trgovine „*Store name*“, a nakon toga se unosi tekst na računu u kojem je navedeno ime trgovine „*Text on bill*“. U polje „*Text on bill*“ unosi se tekst koji skener detektira na računu, stoga je potrebno unijeti točnu cjelovitu liniju teksta kako bi skener uspješno prepoznao trgovinu. Nakon što su unesene vrijednosti u poljima „*Store name*“ i „*Text on bill*“, pritiskom na tipku „*Next*“ prelazi se na sljedeći korak dodavanja trgovine. U ovom koraku, trgovini se dodjeljuje kategorija (Slika 6.9). Izbornik kategorija sortiran je po popularnosti kategorije. Korisnik može odabrati kategoriju klizanjem kroz ponuđene kategorije ili može pretražiti točno određenu kategoriju (Slika 6.10). Kartica odabrane kategorije povećava se u odnosu na druge kartice te se tako naglašava. Pritiskom na gumb „*Add store*“, korisnik se vraća na zaslon na

kojem je bio prije dodavanja trgovine, dodaje se trgovina u bazu podataka te se ispisuje poruka „*Thanks for improving this application*” (Slika 6.11).

Slika 6.8 dodavanje nove trgovine

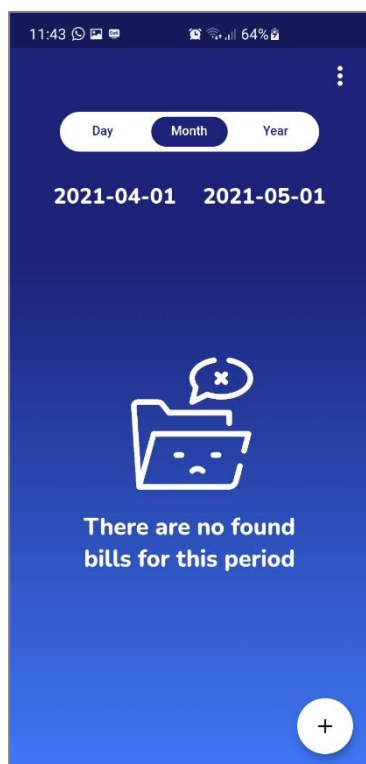
Slika 6.9 odabir kategorije

Slika 6.10 pretraživanje kategorije

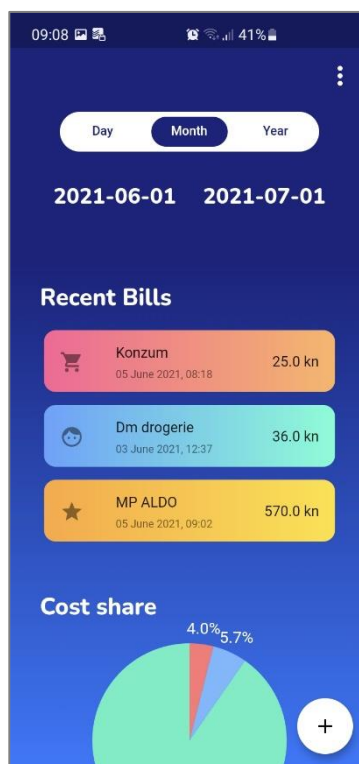
Slika 6.11 poruka zahvale

6.4. Pregledavanje zapisa računa

Na početnoj stranici dostupan je pregled svih zapisa računa za određeni period. Period može biti dan, mjesec ili godina, a zadaje se odabirom opcije u selektoru vremenskog perioda (Slika 5.2). Klizanjem desno-lijevo po zaslonu, period se pomiče za jednu jedinicu perioda (npr. ako je izabran period mjesec u rasponu 1. svibanj – 1.lipanj, pomakom u lijevo raspon postaje 1. travanj – 1. svibanj). Ako za određeni period ne postoje zapisi, na ekranu se prikazuje poruka „*There are no found bills for this period*“ (Slika 6.12), u suprotnome prikazuje se klizeća lista raznovrsnih pogleda na podatke (Slika 6.13).



Slika 6.12 period za koji
nema zapisa



Slika 6.13 period za koji
postoje zapisi

Trenutno dostupni pogledi na zapise su:

- Lista koja prikazuje sve zapise za određeni period. Pritiskom na zapis otvara zaslon za uređivanje zapisa (Slika 6.16).
- Kružni graf koji pokazuje raspodjelu troškova po trgovinama. Pritiskom na dijagram, izmjenjuje se prikaz za grupiranje troškova po trgovinama (Slika 6.14) ili po kategorijama (Slika 6.15).
- Stupčasti dijagram koji prikazuje troškove kroz odabrani period (Slika 6.17).



Slika 6.14 kružni dijagram raspodijeljen po trgovinama



Slika 6.15 kružni dijagram raspodijeljen po kategorijama



Slika 6.16 lista zapisa računa





Slika 6.17 stupčasti dijagram

6.5. Pregledavanje i uređivanje zapisa računa

Aplikacija nudi mogućnost pregledavanja i uređivanja zapisa računa. Odabirom zapisa s liste, otvara se zaslon u kojem aplikacija nudi uređivanje i detaljniji pregled zapisa (Slika 6.18).

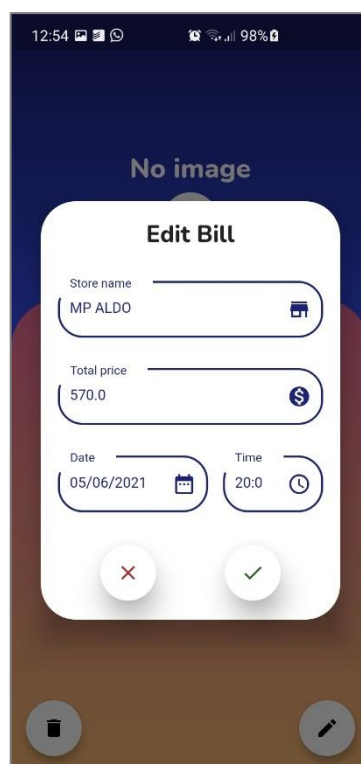
6.5.1. Uređivanje zapisa računa

Pritiskom na gumb , na zaslonu se prikazuje lebdeći prozor u kojem se može uređivati zapis (Slika 6.19). Moguće je promijeniti naziv trgovine, cijenu, datum i vrijeme zapisa. Pri promjeni datuma i vremena, prikazuje se posebna tipkovnica za datum i vrijeme (Slika 6.20).

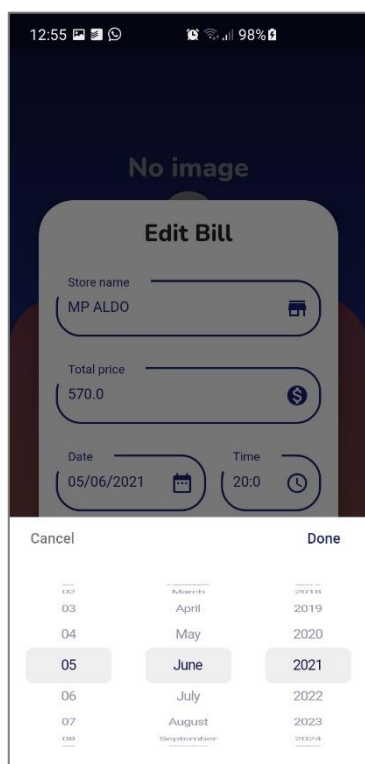
Pritiskom na gumb  moguće je obrisati zapis (Slika 6.21)



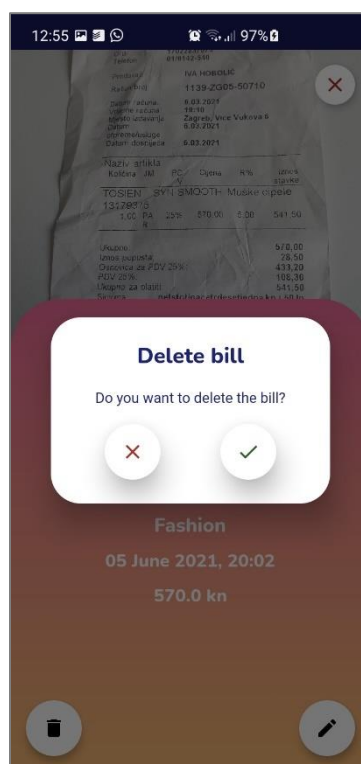
Slika 6.18 pregled zapisa



Slika 6.19 uređivanje zapisa





Slika 6.20 tipkovnica za datum



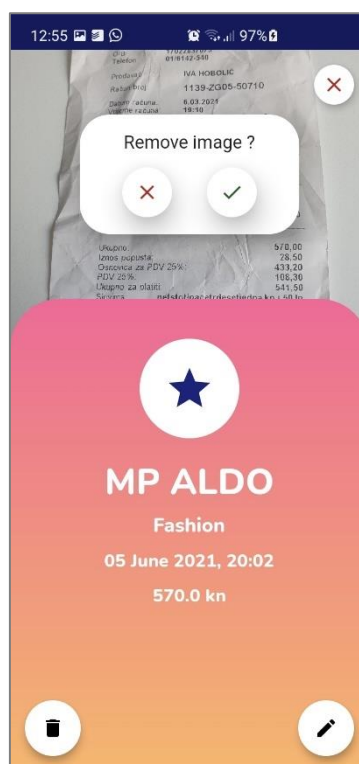
Slika 6.21 brisanje zapisa

6.5.2. Dodavanje fotografije računa

Tijekom uređivanja zapisa, moguće je dodati fotografiju računa. Pritiskom na gumb  (Slika 6.18) otvara se prozor za slikanje. Nakon slikanja i potvrde slike, ona se sprema u memoriju uređaja te se prikazuje u gornjem dijelu ekrana prilikom pregledavanja zapisa (Slika 6.22). Fotografiju je moguće otvoriti u punom zaslonu (engl. *full screen*). Pritiskom na gumb  moguće je ukloniti fotografiju (Slika 6.23).




Slika 6.22 prikaz fotografije računa



Slika 6.23 uklanjanje fotografije

6.6. Dohvaćanje podataka u XLSX formatu

Aplikacija nudi izvoz podataka u XLSX formatu. Na početnom zaslonu, pritiskom na dodatne opcije () nudi se opcija „Export to Excel“. Pritiskom na taj gumb, korisniku se nudi izbor načina na koji želi podijeliti datoteku.

Zaključak

Potreba za praćenjem troškova u današnjem potrošačkom društvu postala je iznimno velika. Kupovina i prodaja odvija se u svakodnevnim situacijama, primjerice u prometu, ugostiteljskim objektima, dućanima i slično. U takvim prilikama, korisnicima je potrebno prenosivo i jednostavno rješenje u vidu mobilne aplikacije koja omogućuje praćenje troškova na brz i efikasan način. Na današnje aplikacije postavljeni su brojni zahtjevi kvalitete i performansi, a optimalno rješenje obično zahtijeva korištenje najmodernijih tehnologija i postupaka.

U okviru ovoga rada razvijen je mobilni programski proizvod namijenjen za praćenje, evidentiranje i kontrolu troškova. Aplikacija na inovativan način omogućava skeniranje i prepoznavanje dijelova računa korištenjem tehnologija iz područja računalnog vida. Aplikacija ima potencijal za široku primjenu jer u današnje vrijeme veliki broj ljudi posjeduje mobilni uređaj na kojem mogu jednostavno instalirati mobilnu aplikaciju i koristiti ju u pokretu.

Aplikacija bi se mogla dodatno unaprijediti primjenom strojnog učenja koje bi omogućilo predviđanje lokacija pojedinih dijelova teksta na računu, automatsko dodavanje novih računa i prepoznavanje lokaliziranih postavki. Također, korisnički podaci bi se mogli spremati u bazu podataka i tako omogućiti udruživanje korisnika (npr. u istom kućanstvu) te zajedničko praćenje troškova. Još jedan dodatak aplikaciji bila bi mogućnost postavljanja dnevnih, mjesečnih i godišnjih financijskih ciljeva i ograničenja na čije bi prekoračenja aplikacija upozoravala korisnika.

Literatura

- [1] PixelRush, *IMoney*, GooglePlay.
https://play.google.com/store/apps/details?id=org.pixelrush.moneyiq&utm_source=official_site&pcampaignid=pcampaignidMKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1; posjećeno 23.5.2021.
- [2] Realbyte Inc., *Money Manager*, GooglePlay.
Poveznica <https://play.google.com/store/apps/details?id=com.realbyteapps.moneymanagerfree&hl=hr&gl=US>; posjećeno 23.5.2021.
- [3] Standy Software, *Snap & Split Bill*, GooglePlay.
<https://play.google.com/store/apps/details?id=com.astepanov.mobile.splitcheck&hl=hr&gl=US>; posjećeno 24.5.2021.
- [4] Gaël Thomas, *What is Flutter and Why You Should Learn it in 2020*, freeCodeCamp (prosinac 2019). : <https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/>; posjećeno 21.5.2021.
- [5] *The Good and the Bad of Firebase Backend Services*, Altexsoft.
<https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/>; posjećeno 26.6.2021.
- [6] *Cloud Firestore*, Firebase. <https://firebase.google.com/products/firestore>
- [7] *Cloud Firestore*, FlutterFire. <https://firebase.flutter.dev/docs/firestore/usage/>.
- [8] Pub.dev. https://pub.dev/packages/flutter_camera_ml_vision
- [9] *Firebase Machine Learning*, Firebase. <https://firebase.google.com/products/ml>
- [10] *Introduction to widgets*, Flutter. <https://flutter.dev/docs/development/ui/widgets-intro>
- [11] Pub.dev. <https://pub.dev/packages/diacritic>

Sažetak

Mobilna aplikacija za evidenciju troškova

Praćenje financija vrlo je važan zadatak s kojim se danas susreće većina ljudi. Aplikacije za praćenje troškova omogućuju obavljanje te zadaće na jednostavniji, brži i praktičniji način, pružajući mobilnost i fleksibilnost korištenja.

Mobilna aplikacija „Scan Bill“ razvijena u sklopu ovoga rada nudi mogućnost praćenja, evidentiranja i izvoza podataka u Excel formatu. Programski proizvod omogućuje skeniranje računa pomoću kamere mobilnog uređaja korištenjem postupka optičkog prepoznavanja teksta. Također, aplikacija omogućuje ručni unos računa te prikazuje sve troškove, stupčasti i kružni dijagram kako bi korisnik mogao pratiti svoje troškove. Podate o trgovinama unose korisnici aplikacije te se podaci spremaju u Firebase bazu podataka. Aplikacija je razvijena u radnom okviru Flutter te je podržana na operacijskim sustavima Android i iOS.

GitHub link: <https://github.com/FilipPavicic/Final-BSc-Assignment-Zavrzni-rad>

Ključne riječi

Scan Bill, mobilna aplikacija, Flutter, Dart, Android, iOS, Firebase, Firebase alat za strojno učenje, optičko prepoznavanje znakova

Summary

Mobile application for tracking expenses

Tracking expenses is a highly important task for a majority of people. Cost tracking applications allow users to perform this task in a much simpler, faster, and more convenient way by providing mobility and flexibility of use.

"Scan Bill" is a mobile application developed as a part of this assignment. It enables monitoring, documenting, and exporting data in Excel format. This software allows you to scan the bills using a mobile device camera and it is based on optical character recognition. Also, the application allows manual entry of bills and displays all costs, bar and pie chart so that the user can track their expenses. Store information is entered by application users and the data is stored in the Firebase database. The application was developed in the Flutter UI framework and is supported by Android and iOS mobile devices.

GitHub link: <https://github.com/FilipPavicic/Final-BSc-Assignment-Zavrsni-rad>

Keywords

Scan Bill, mobile application, Flutter, Dart, Android, iOS, Firebase, Firebase machine learning kit, optical character recognition