Mobilna aplikacija za praćenje režijskih troškova kućanstva

Lesar, Dominik

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Preddiplomski sveučilišni studij Računarstva

Mobilna aplikacija za praćenje režijskih troškova kućanstva

Završni rad

Dominik Lesar

Osijek, godina. 2023.

SADRŽAJ

1. UVOD

U današnjem načinu života upravljanje financijama postaje sve teže. Kada je pitanje praćenja režijskih troškova kućanstva, pojedinci se suočavaju s praćenjem i razumijevanjem vlastitih financija. Zbog tog razloga aplikacije za praćenje troškova postaju sve popularnije, te olakšavaju taj proces.

Ovaj završni rad će se ogledati na razvoj jedne takve mobilne aplikacije koja će omogućiti korisnicima jednostavno praćenje režijskih troškova kućanstva te ostalih troškova. Android aplikacija će koristiti Firebase Firestore [1] kao bazu podataka za spremanje raznih podataka vezano uz pojedini trošak. Jedna od funkcionalnosti ove aplikacije je skeniranje PDF417 linijskog koda [2], s raznih računa za režije kućanstva pomoću Google ML kit biblioteke [3]. Za prikazivanje podataka o troškovima unutar aplikacije koristit će se MPAndroidChart [4] biblioteka koja služi za prikazivanje podataka pomoću raznih grafova i dijagrama.

U narednom dijelu završnog rada, razmotrit ćemo njegovu strukturu po poglavljima. U drugom poglavlju opisane su slične aplikacije. Treće poglavlje opisuje arhitekturu aplikacije, korištene alate i dizajn korisničkog sučelja. U četvrtom poglavlju opisan je programerski kod aplikacije. Peto poglavlje prikazuje tijek testiranja aplikacije kroz opis i slike. Na kraju završnog rada nalazi se zaključak.

1.1. Zadatak završnog rada

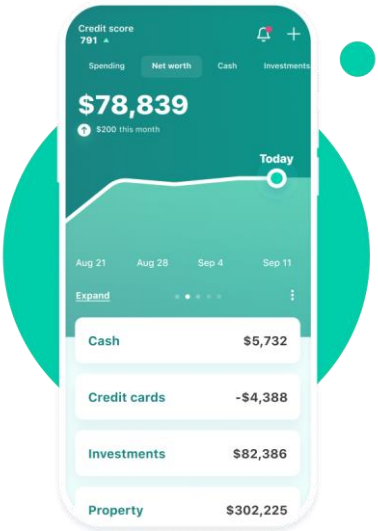
Zadatak ovog završnog rada je razviti mobilnu Android aplikaciju za praćenje režijskih troškova kućanstva. Aplikacija će moći skenirati račun za režije te automatski unijeti potrebne podatke s računa, ali će imati i mogućnost ručnog unošenja podataka o trošku. Također, u aplikaciji će biti jednostavan prikaz troškova u obliku liste.

2. SLIČNE APLIKACIJE

Razvojem tehnologije i učestalom mijenjanju cijena nastala je potreba za alatom pomoću kojeg osoba na jednom mjestu može bilježiti i pratiti svoje troškove i financije. Na tržištu postoji mnogo aplikacija za praćenje troškova, a u ovom poglavlju su navedene neke od njih.

2.1. Mint

Mint je popularna mobilna aplikacija koja omogućuje svojim korisnicima praćenje osobnih financija, uključujući i režijske troškove [5]. Uvid u troškove korisnicima prikazuje pomoću grafova te aplikacija sama kategorizira troškove (Slika 2.1.). Podržava integraciju s različitim bankovnim računima. Podržava postavljanje financijskih ciljeva i praćenja investiranja, te ima opciju podsjetnika za plaćanje računa. U odnosu na izrađenu aplikaciju Mint aplikacija ima mogućnost grafičkog prikaza podataka o troškovima u obliku linijskih grafova.



Sl. 2.1. Sučelje Mint aplikacije

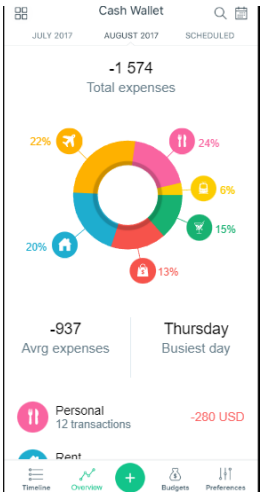
2.2. Wallet

Wallet je mobilna aplikacija za praćenje troškova koja omogućuje korisnicima ručno unošenje troškova, ali i automatsko unošenje putem integracije s bankovnim računom [6]. Podržava povezivanje kartice te brzo plaćanje. Korisničko sučelje je jednostavno i funkcionalno osiguravajući korisnicima jednostavno praćenje svojih financija (Slika 2.2.). Pri usporedbi s izrađenom aplikacijom korisničko sučelje Wallet aplikacije je intuitivnije i funkcionalnije.

Sl. 2.2. Sučelje Wallet aplikacije

2.3. Spendee

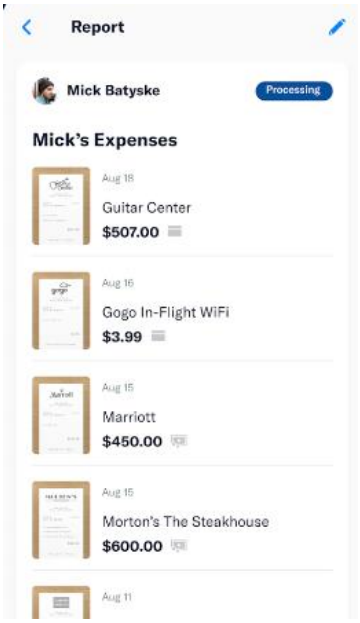
Spendee je jednostavna mobilna aplikacija koja omogućuje unos i kategorizaciju troškova [7]. Prikazuje podatke o troškovima pomoću grafova (Slika 2.3.). Olakšava štednju i praćenje troškova. Jedna od prednosti u usporedbi s izrađenom aplikacijom je postavljanje financijskih ciljeva.



Sl. 2.3. Sučelje Spendee aplikacije

2.4. Expensify

Expensify je mobilna aplikacija dizajnirana za jednostavnije upravljanje troškovima pojedinca ili tvrtke. Nudi detaljno generirana izvješća o troškovima. Podržava integraciju s kreditnim karticama i bankovnim računima, te automatsko skeniranje računa. Prednost ove aplikacije u usporedbi s izrađenom aplikacijom je spremanje fotografija računa uz ostale podatke (Slika 2.4.).



Sl. 2.4. Sučelje Expensify aplikacije

2.5. Monefy

Monefy je aplikacija za upravljanje financijama putem raznih alata za praćenje prihoda, troškova i ulaganja. Korisnici mogu izraditi detaljne proračune, pratiti trendove potrošnje i precizno postaviti financijske ciljeve. Aplikacija ima napredne mogućnosti analitike i izvješćivanja kako bi korisnici mogli što lakše donijeti financijsku odluku. Jedna od prednosti Monefy aplikacije u odnosu na izrađenu aplikaciju je jednostavnije dodavanje novih troškova.

3. ARHITEKTURA APLIKACIJE

Aplikacija je pisana u Kotlin programskom jeziku. Korisnici mogu unositi režijske troškove koji se spremaju na bazu podataka. Podaci o troškovima prikazani su u obliku kružnog dijagrama. Korisnik ima opciju unijeti određeni trošak ručno ili pomoću skeniranja PDF417 linijskog koda pomoću kamere na Android uređaju. Ovo poglavlje se sastoji od detaljnijeg opisa korištenih tehnologija i alata, baze podataka i dizajna korisničkog sučelja.

3.1. Korištene tehnologije i alati

Za pohranu podataka koristi se Firebase Firestore baza podataka. Za skeniranje linijskog koda s računa koristi se Google ML Kit biblioteka. Jednostavan prikaz podataka u obliku kružnog dijagrama na glavnoj stranici aplikacije se vrši pomoću MPAndroidChart biblioteke. Za provjeru korisničkih podataka koristi se Firebase Authentication.

3.1.1. Kotlin programski jezik

Kotlin [8] je višeplatformski, statički tipiziran programski jezik visoke razine koji se koristi za razvoj Android aplikacija. Kotlin je dizajniran da međusobno sudjeluje s Java programskim jezikom. Kotlin i Java pokreću se na Java virtualnom stroju što omogućuje da se međusobno mogu lako pozivati. Kotlin posjeduje značajke koje razvojnim programerima omogućuju brži i sigurniji razvoj [9]. Kotlin se još uvijek najviše koristi za razvoj aplikacija na Androidu, a Google procjenjuje da je 70% od 1000 najpopularnijih aplikacija u trgovini Google Play napisano u Kotlinu. Također, Kotlin je projekt otvorenog koda koji je dostupan bez naknade pod licencom Apache 2.0. Kôd za projekt razvija prvenstveno tim zaposlen u JetBrains razvojnoj tvrtki, uz doprinose Googlea i drugih.

3.1.2. Android studio

Android studio [10] je službeno razvojno okruženje za razvoj Android aplikacija izgrađen na JetBrainsovom IntelliJ IDEA softveru i dizajniran posebno za razvoj aplikacija za Android platformu. Sadrži bogati uređivač izgleda koji korisnicima omogućuje povlačenje i ispuštanje komponenti korisničkog sučelja, mogućnost pregleda izgleda na više konfiguracija zaslona,

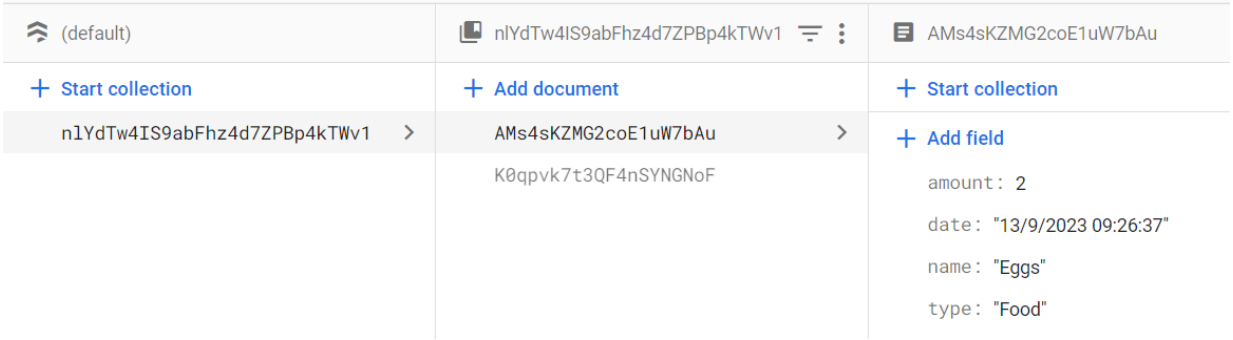
automatsko dovršavanje koda, upravljanje verzijama aplikacije, te razne alate za otklanjanje grešaka. Također, sadrži Lint alate za identificiranje performansi, upotrebljivosti, kompatibilnosti verzija i drugih problema.

3.1.3. Firebase Cloud Firestore

Firebase je skup pozadinskih usluga računalstva u oblaku i platformi za razvoj aplikacija koje pruža Google. Cloud Firestore je fleksibilna, skalabilna baza podataka za razvoj mobilnih, web i serverskih aplikacija. Sinkronizira podatke u svim klijentskim aplikacijama putem slušatelja u stvarnom vremenu i nudi izvanmrežnu podršku za mobilne uređaje. Lokalno sprema podatke koje aplikacija aktivno koristi, tako da aplikacija može pisati, čitati, primati i postavljati upite za podatke kada je uređaj izvan mreže. Kada se uređaj vrati na mrežu, Cloud Firestore sinkronizira sve lokalne promjene natrag u Cloud Firestore. Cloud Firestore je NoSQL baza podataka smještena u oblaku kojoj Apple, Android i web aplikacije mogu pristupiti izravno putem izvornih SDK-ova. Sinkronizira podatke u svim klijentskim aplikacijama putem slušatelja u stvarnom vremenu i nudi izvanmrežnu podršku za mobilne uređaje i web kako bi se mogle izraditi responzivne aplikacije koje rade bez obzira na latenciju mreže ili internetsku povezanost. Cloud Firestore također nudi besprijekornu integraciju s drugim Firebase i Google Cloud proizvodima, uključujući Cloud Functions.

3.2. Arhitektura baze podataka

Firebase Cloud Firestore je dokumentno-orijentirana baza podataka organizirana u obliku stabla JSON. Svaki novi registrirani korisnik u bazi dobije svoju kolekciju u koju se spremaju njegovi troškovi. Svaka kolekcija ima unikatni identifikacijski broj koji se pridoda korisniku prilikom registracije.



Sl. 3.1. Baza podataka

Nakon što korisnik u aplikaciji doda novi trošak, u kolekciju od korisnika na bazi podataka sprema se dokument s podacima o trošku (iznos, vrijeme dodavanja troška, ime troška, i tip troška).

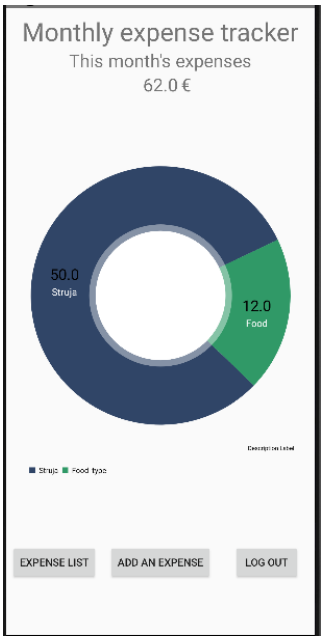
Na slici 3.1. je prikazana kolekcija kojoj je naziv po unikatnom korisničkom broju koji korisnik dobije prilikom registracije. Unutar te kolekcije su dva dokumenta koja se odnose na dva troška koja je korisnik dodao. Svaki trošak ima iznos, datum unošenja, ime i tip.

3.3. Dizajn korisničkog sučelja

U ovom dijelu završnog rada opisan je dizajn korisničkog sučelja. Sučelje je jednostavno s prikazom mjesečnog troška na glavnoj aktivnosti pomoću kružnog dijagrama. Kroz aplikaciju se navigira putem gumbova koji otvaraju nove aktivnosti. Unos podataka se radi kroz jednostavna polja za unos teksta.

3.3.1. Početna aktivnost

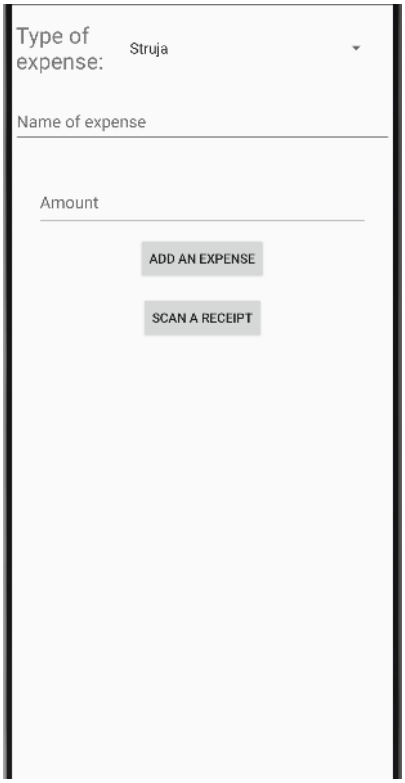
Početna aktivnost prikazana na slici 3.2. sastoji se od grafa napravljenog pomoću MPAndroidChart biblioteke, koji prikazuje troškove na kružnom dijagramu po tipu troška za tekući mjesec, te na vrhu ukupan zbroj troškova za taj mjesec. Na aktivnosti se nalaze tri gumba, jedan za prikaz troškova u listi po mjesecu, jedan za dodavanje troška i treći za odjavljivanje. Gumb za prikaz troškova u listi otvara novu aktivnost u kojoj se nalazi lista troškova s njihovim podacima ovisno o odabranom mjesecu. Gumb za dodavanje troška otvara aktivnost za dodavanje troška kojeg se može unijeti ručno ili putem skeniranja PDF417 linijskog koda.



Sl. 3.2. Početna aktivnost

3.3.2. Aktivnost za dodavanje troška

Na aktivnosti za dodavanje troška (Slika 3.3.) se može odabrati tip, unijeti ime troška, te iznos, ručno, ili automatski aktivacijom gumba „Scan a receipt“ nakon čega se otvori kamera te je potrebno skenirati PDF417 linijski kod s računa. PDF417 je naslagani (sastoji se od više linearnih linijskih kodova naslaganih jedni na druge) linearni format linijskog koda koji ima primjenu u različitim aplikacijama kao što su transport, identifikacijske kartice i upravljanje inventarom. "PDF" je skraćenica za Portable Data File. "417" označava da se svaki uzorak u kodu sastoji od 4 crte i razmaka u uzorku koji je dug 17 jedinica (modula).



Sl. 3.3. Aktivnost za dodavanje troška

3.3.3. Aktivnost za prikaz liste troškova

Na aktivnosti za prikaz troškova (Slika 3.4.) nalaze se troškovi ovisno o mjesecu kojeg smo izabrali. Prikaz troškova u ovom formatu se vrši pomoću Android studio komponente *Recycler View*. Odabir mjeseca se vrši pomoću padajućeg izbornika na dnu aktivnosti.

Sl. 3.4. Aktivnost za prikaz troškova

4. IMPLEMENTACIJA

U ovom poglavlju će biti opisani koraci u kreiranju aplikacije uz objašnjenje programskog koda. Ključni koncepti ove Android aplikacije su prikazivanje grafa na početnoj aktivnosti pomoću MPAndroidChart biblioteke, te skeniranje linijskog koda pomoću Google ML Kit biblioteke. Podaci se spremaju i dohvaćaju pomoću Firebase Cloud Firestore baze podataka. Aplikacija je implementirana pomoću programskog jezika Kotlin u razvojnom okruženju Android Studio.

4.1. Biblioteke

Prije same implementacije potrebno je u projekt uključiti biblioteke nužne za daljnji razvoj aplikacije (Slika 4.1.). Neke od ključnih biblioteka uključenih u aplikaciju su: MPAndroidChart, Google ML Kit, CameraX, Firebase Cloud Firestore. CameraX biblioteka služi za rad s kamerom unutar aplikacije koja šalje podatke Google ML Kit na analizu. MPAndroidChart je biblioteka za prikaz podataka pomoću raznih grafova poput linijskih grafova, stupčastih grafova, dijagrama raspršenja. U aplikaciji se koristi kružni dijagram koji je dio te biblioteke. Google ML kit je skup alata koji koriste strojno učenje za razna rješenja poput skeniranja linijskih kodova, prepoznavanja lica i prepoznavanja teksta. Za potrebu izrade aplikacije korišteno je već implementirano rješenje za skeniranje linijskih kodova unutar Google ML kit biblioteke.

4.2. Provjera korisničkih podataka

Kako bi se podaci spremali u bazu podataka za svakog pojedinog korisnika potrebno je napraviti provjeru korisničkih podataka. Prilikom registracije svaki korisnik dobije unikatni korisnički broj koji se koristi za spremanje podataka u bazu tog određenog korisnika. Svaka kolekcija nazvana po unikatnom korisničkom broju korisnika sadrži sve troškove tog korisnika. Provjera korisničkih podataka je u aplikaciji implementirana pomoću Firebase authentication biblioteke. Na ovaj način korisnici mogu pristupati svojim troškovima neovisno o uređaju na kojem pokreću aplikaciju. Na slici 4.1. prikazan je isječak koda koji služi za provjeru korisničkih podataka.

Nakon pritiska gumba „Login“ pokreće se funkcija *login()* koja pomoću Firebase authentication biblioteke prijavljuje korisnika ukoliko je već registriran, te se preusmjerava na početnu aktivnost gdje se prikazuju podaci o njegovim troškovima. Funkcija *login()* će se pokrenuti samo ako polja za unos nisu prazna, to vrijedi i za aktivnost za registraciju. Nakon što se zahtjev pošalje na

Firebase obavlja se provjera korisničkih podataka i ukoliko je korisnik točno upisao adresu elektroničke pošte i zaporku otvara se glavna aktivnost.

```
private fun login() {
    val email = etEmail.text.toString()
    val pass = etPass.text.toString()
    auth.signInWithEmailAndPassword(email, pass).addOnCompleteListener(this) { it: Task<AuthResult!> ->
        if (it.isSuccessful) {
            Toast.makeText( context: this,  text: "Successfully LoggedIn", Toast.LENGTH_SHORT).show()
            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
            finish()
        } else
            Toast.makeText( context: this,  text: "Log In failed ", Toast.LENGTH_SHORT).show()
    }
}
```

Sl. 4.1. Isječak koda za prijavu korisnika

4.3. Prikaz troškova na glavnoj stranici

Na glavnoj aktivnosti aplikacije prikazuje se kružni dijagram pomoću biblioteke MPAndroidChart koji vizualno prikazuje količinu mjesečnih troškova ovisno o tipu troška, te ukupan trošak za taj mjesec.

```
var currentMonthYear = SimpleDateFormat( pattern: "MM/yyyy").format(System.currentTimeMillis())
val db = FirebaseFirestore.getInstance()
val collectionRef = db.collection(FirebaseAuth.getInstance().uid.toString())
collectionRef.get()
    .addOnSuccessListener { documents ->
        Log.d( tag: "Main", msg: "CALLED SUCCESS")
        var totalExpense = 0f
        for (document in documents) {
            val type = document.getString( field: "type") ?: ""
            val dateString = document.getString( field: "date") ?: ""
            val documentMonthYear = SimpleDateFormat( pattern: "dd/MM/yyyy HH:mm:ss").parse(dateString)
            val documentFormattedMonthYear = SimpleDateFormat( pattern: "MM/yyyy").format(documentMonthYear)


            if (currentMonthYear == documentFormattedMonthYear) {
                if (!types.contains(type)) {
                    types.add(type)
                }
                val amount = document.getDouble( field: "amount")?.toFloat() ?: 0f
                totalExpense += amount
                expenses[type] = (expenses[type] ?: 0f) + amount
            }
        }
        if(types.isNotEmpty()){
            for (type in types) {
                val typeValueSum = expenses[type] ?: 0f
                pieEntries.add(PieEntry(typeValueSum, type))
            }
        }
```

Sl. 4.2. Isječak koda za dohvaćanje podataka o troškovima

Na slici 4.2. prikazan je kod koji nakon dohvaćanja dokumenata s baze podataka prolazi kroz sve dohvaćene dokumente u kojima su pohranjeni podaci o određenom trošku, svaki dokument sadrži podatke o jednom trošku. Datumi se formatiraju kako bi se mogli usporediti sa sadašnjim datumom, na taj način na kružnom dijagramu se prikazuju samo troškovi trenutnog mjeseca. Zbraja se ukupni trošak tog mjeseca te se prikazuje na vrhu glavne aktivnosti. Nakon toga se u listu *pieEntries* spremaju podaci o sumi iznosa tog tipa troška i tip koji se prikazuju na glavnoj aktivnosti.

```
val colors = ArrayList<Int>()
colors.add(Color.parseColor( colorString: "#304567"))
colors.add(Color.parseColor( colorString: "#309967"))
colors.add(Color.parseColor( colorString: "#476567"))
colors.add(Color.parseColor( colorString: "#890567"))
colors.add(Color.parseColor( colorString: "#a35567"))
colors.add(Color.parseColor( colorString: "#ff5f67"))
colors.add(Color.parseColor( colorString: "#3ca567"))

val pieDataSet = PieDataSet(pieEntries, label)
pieDataSet.valueTextSize = 20f
pieDataSet.colors = colors

val pieData = PieData(pieDataSet)
pieData.setDrawValues(true)

findViewById<TextView>(R.id.totalExpenseValue).text = totalExpense.toString()
pieChart.data = pieData
pieChart.invalidate()
```

Sl. 4.3. Isječak koda za dohvaćanje podataka o troškovima

Na slici 4.3. prikazan je isječak koda zaslužan za prikaz podataka o troškovima. U listu *colors* dodaju se boje s kojima će se prikazivati tipovi troškova na kružnom dijagramu i njihov ukupan iznos. Stvoren je *PieDataSet* koji definira kako bi kružni dijagram trebao izgledati. Inicijalizira se s *pieEntries*, koji sadrži podatke za svaki dio kružnog dijagrama. Ukupan trošak se prikazuje u tekstnom okviru na vrhu te se na samom kraju pomoću *pieChart.invalidate()* osvježi i ponovno iscrta kružni dijagram s ažuriranim podacima. Svaki tip troška ovisno o njegovom iznosu zauzima određeni dio kružnog dijagrama, taj dio dobije jednu od predefiniranih boja iz liste *colors*. Na taj način korisnik dobije vizualni pregled troškova za taj mjesec.

4.4. Dodavanje troška

Na aktivnosti za dodavanje troška nalazi se padajući izbornik s tipovima troška, polje za unos imena troška te polje za unos iznosa troška. Padajući izbornik sadrži tipove troškova koji su već uneseni tako da se dohvate svi troškovi korisnika iz baze. Prolazi se kroz troškove i ukoliko tip određenog troška nije u listi *types* dodaje se u tu listu (Slika 4.4.).

```
collectionRef.get()
    .addOnSuccessListener { documents ->
        val types = ArrayList<String>()
        for (document in documents) {
            val type = document.getString( field: "type") ?: ""
            if (!types.contains(type)) {
                types.add(type)
            }
        }
        types.add(types.size,  element: "Add a new type")
        val typeSpinner = findViewById<Spinner>(R.id.typeSpinner)
        val spinnerAdapter = CustomSpinnerAdapter( context: this, android.R.layout.simple_spinner_item, types)
        spinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        typeSpinner.adapter = spinnerAdapter

        typeSpinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
                val selectedOption = types[position]
                if (selectedOption == "Add a new type") {
                    showAddTypeDialog(types, spinnerAdapter)
                }

                Log.d( tag: "Spinner",  msg: "Selected option: $selectedOption")
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {
            }
        }
```

Sl. 4.4. Isječak koda za dohvaćanje podataka o troškovima

Kada se prođe kroz sve troškove i dodaju svi tipovi, na kraj liste se dodaje element „Add a new type“ koji, ako se odabere, poziva funkciju *showAddTypeDialog* (Slika 4.5.) koja otvara dijalog za dodavanje novog tipa troška. Unutar dijaloga moguće je dodati novi tip troška, koji se dodaje u padajući izbornik nakon aktivacije gumba „Add new expense type“. Na ovaj način implementirano je jednostavno dodavanje novih tipova troškova.

```
private fun showAddTypeDialog(types: ArrayList<String>, adapter: ArrayAdapter<String>) {
    val dialog = Dialog( context: this)
    dialog.setContentView(R.layout.dialog_add_type)

    val editText = dialog.findViewById<EditText>(R.id.editTextType)
    val addButton = dialog.findViewById<Button>(R.id.addButtonDialog)

    addButton.setOnClickListener { it: View!
        val newType = editText.text.toString()
        if (newType.isNotEmpty()) {
            types.add( index: types.size - 1, newType)
            adapter.notifyDataSetChanged()
            dialog.dismiss()
        } else {
            Toast.makeText( context: this,  text: "Please enter a type", Toast.LENGTH_SHORT).show()
        }
    }

    dialog.show()
}
```

Sl. 4.5. Isječak koda za prikazivanje dijaloga

Nakon pritiska gumba „Add Expense“ svi podaci iz polja za unos teksta i iz padajućeg izbornika se spremaju u varijable koje se mapiraju u *hashMap* tip podatka koji se može zapisati u bazu podataka (Slika 4.6.).

```
addButton.setOnClickListener { it: View!
    if(amountTextInput.text?.isNotEmpty() == true || nameOfExpense.text?.isNotEmpty() == true){
        val intent = Intent( packageContext: this, MainActivity::class.java)
        val amount = amountTextInput.text.toString().toFloat()
        val type = typeSpinner.selectedItem.toString()
        val name =  nameOfExpense.text.toString()
        val sdf = SimpleDateFormat( pattern: "dd/M/yyyy hh:mm:ss")
        val data = hashMapOf("amount" to amount, "type" to type, "name" to name, "date" to sdf.format(
            Date()))

        collectionRef.document() DocumentReference
            .set(data) Task<Void!>
            .addOnSuccessListener { it: Void!
                Toast.makeText( context: this,  text: "Data updated successfully!", Toast.LENGTH_SHORT).show()
                startActivity(intent)
                finish()
            }
            .addOnFailureListener { e ->
                Toast.makeText( context: this,  text: "Error updating data: $e", Toast.LENGTH_SHORT).show()
            }
    }else{
        Toast.makeText( context: this,  text: "The fields cant be empty.", Toast.LENGTH_SHORT).show()
    }

}
```

Sl. 4.6. Isječak koda za slanje podataka na bazu

4.5. Skeniranje linijskog koda

Skeniranje linijskog koda se vrši pomoću CameraX biblioteke i Google ML Kit biblioteke. Funkcija *analyze* prima parametar tipa *ImageProxy* koji se formatira u oblik pogodan za skeniranje. Kamera u stvarnom vremenu slike šalje analizatoru fotografije koji traži linijski kod na slici pomoću metode *.process()*. Na objekt *scanner* postavljen je slušatelj koji kada aplikacija uspješno skenira linijski kod pokreće aktivnost za dodavanje troška, te joj šalje podatke dohvaćene iz linijskog koda potrebne za automatsko popunjavanje podataka o trošku pomoću *.putExtra()* metode (Slika 4.7.).

```
override fun analyze(image: ImageProxy) {
    val img = image.image
    if (img != null) {
        val inputImage = InputImage.fromMediaImage(img, image.imageInfo.rotationDegrees)

        // Process image searching for barcodes
        val options = BarcodeScannerOptions.Builder()
            .setBarcodeFormats(Barcode.FORMAT_PDF417,
                               Barcode.FORMAT_QR_CODE)
            .build()

        val scanner = BarcodeScanning.getClient(options)

        scanner.process(inputImage)
            .addOnSuccessListener { barcodes ->
                if(barcodes.isNotEmpty()){
                    val firstBarcode = barcodes.first()
                    Log.d( tag: "BARKOD", firstBarcode.rawValue.toString())
                    val intent = Intent(context, ScannedReceiptActivity::class.java)
                    intent.putExtra( name: "barcodeData", firstBarcode.rawValue)
                    context.startActivity(intent)

                }
            }
            .addOnFailureListener { image.close() }

    }
    image.close()
}
```

Sl. 4.7. Isječak koda za analizu barkoda

4.6. Pretvaranje podataka koje sadrži linijski kod u listu

Nakon skeniranja linijskog koda dobije se tekstni niz koji nije primjeren za rukovanje te ga se treba pretvoriti u listu tipa *string* gdje će svaki element liste odgovarati jednoj stavci na računu. U novu listu naziva *returnString*, tipa *string*, dodaju se samo dvije stavke sa skeniranog naloga, ime troška i iznos. Nakon uspješnog skeniranja poziva se funkcija *barcodeDataToList()* koja prima parametar *barcodeData* tipa *string* koji se dobije nakon skeniranja (Slika 4.8.).

```
private fun barcodeDataToList(barcodeData : String) : ArrayList<String>{
    val returnList : ArrayList<String> = ArrayList()
    val list = barcodeData.lines()
    Log.d( tag: "EX",  msg: "LIST: " + list)
    Log.d( tag: "EX", list[list.lastIndex-1])
    val sb = StringBuilder(list[2])
    sb.insert( offset: list[2].count() - 2,  str: ".")

    val value = sb.toString().toDouble()
    returnList.add(value.toString())
    val desc = list[list.lastIndex-1]
    returnList.add(desc)
    return returnList
}
```

Sl. 4.8. Isječak koda koji prikazuje funkciju *barcodeDataToList()*

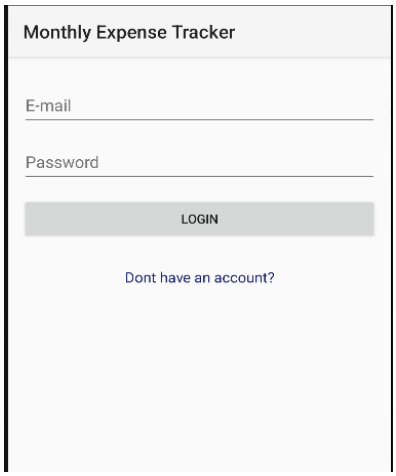
Podaci se dobiju u obliku tekstnog niza gdje je svaka stavka s naloga za nacionalna plaćanja u posebnom redu (Slika 4.9.). Pomoću metode *.lines()* se podaci pretvore u listu podataka tako da svaki red iz tekstnog niza postane jedan element liste radi lakšeg rukovanja. Nakon toga se formatira iznos računa. Opis i formatirani iznos se dodaju u listu čiji se elementi automatski postavljaju u pripadna polja unutar aktivnosti za dodavanje novog troška.

```
HRVHUB30
EUR
000000000000742
Ime Prezime
Ulica  185
32281 Grad
HEP
Lj. Gaja 1c
32100 Vinkovci
HR5224810001519000062
HR01
210090185-230446699
COST
Racun za 10 i 12/2023.god.
```

Sl. 4.9. Struktura podataka dobivenih nakon skeniranja računa

5. TESTIRANJE APLIKACIJE

Pri ulasku u aplikaciju prikaže se aktivnost za prijavu (Slika 5.1.). Ukoliko je netko novi korisnik može pritisnuti gumb za registraciju, te napraviti novi račun. Provjera korisničkih podataka je implementirana pomoću Firebase authentication biblioteke. Ukoliko se korisnik pokuša prijaviti s krivim podacima aplikacija će prikazati upozorenje, također ako korisnik ostavi polja za unos elektroničke pošte i zaporke praznima i pokuša se prijaviti aplikacija će upozoriti na to.

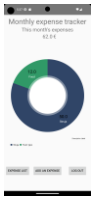


Sl. 5.1. Stranica za prijavu

Ukoliko korisnik aktivira gumb „Dont have an account?“ otvori se aktivnost za registraciju novog korisnika (Slika 5.2.). Sva polja moraju biti popunjena i zaporke se moraju podudarati. Također, ako zaporka ima manje od 8 znakova i nije mješavina slova i brojeva korisnik se neće moći registrirati i aplikacija će upozoriti na to. Prilikom registracije korisnik će dobiti unikatni identifikacijski broj pomoću kojega se spremaju podaci u bazu podataka.

Sl. 5.2. Stranica za registraciju

Nakon uspješne autentifikacije dolazi se na glavnu aktivnost (Slika 5.3.) gdje su prikazani troškovi tekućeg mjeseca u obliku kružnog dijagrama. Dijagram prikazuje ukupan zbroj troškova prema tipu u tekućem mjesecu, svaki tip je označen drugačijom bojom. Ukupan zbroj troškova za tekući mjesec je prikazan na vrhu početne aktivnosti. Na glavnoj aktivnosti također postoji gumb „Log out“ koji pri aktivaciji odjavljuje korisnika iz računa. Korisnik ostaje prijavljen i nakon izlaza iz aplikacije sve dok ne dođe do aktivacije gumba „Log out“, što pruža brz i jednostavan pristup podacima o troškovima.

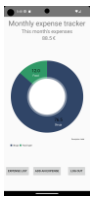


Sl. 5.3. Glavna stranica

Novi trošak se može dodati aktivacijom dugmeta „Add an expense“, nakon čega se otvori nova aktivnost za dodavanje troška (Slika 5.4.). Trošak se može unijeti ručno ili pomoću skeniranja PDF417 linijskog koda pritiskom na gumb „Scan a receipt“. Sva polja moraju biti popunjena u protivnom će se prikazati greška. Pritiskom na padajući izbornik može se odrediti tip troška, te dodati novi pritiskom na opciju „Add new type“.

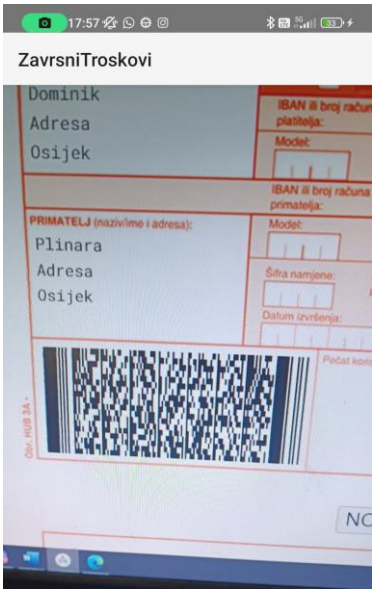
Sl. 5.4. Stranica za dodavanje troška

Aktivacijom dugmeta „Add an expense“ podaci za trošak šalju se u bazu podataka, te se ponovno otvara glavna aktivnost i prikazuju ažurirani podaci (Slika 5.5.). Prednost korištenja baze podataka je mogućnost pristupanja troškovima neovisno o uređaju koji pokreće aplikaciju, jer su svi podaci o troškovima spremljeni u bazi podataka i korisnik im može pristupiti prijavom na njegov račun.



Sl. 5.5. Glavna stranica nakon dodavanja troška

Skeniranje računa je moguće aktivacijom dugmeta „Scan a receipt“ na stranici za dodavanje troška prikazanoj na slici 5.4., nakon toga se otvori kamera unutar aplikacije (Slika 5.6.), te je moguće skenirati linijski kod. Za otvaranje kamere unutar aplikacije koristi se biblioteka CameraX koja u stvarnom vremenu šalje slike Google ML kit čitaču linijskih kodova.

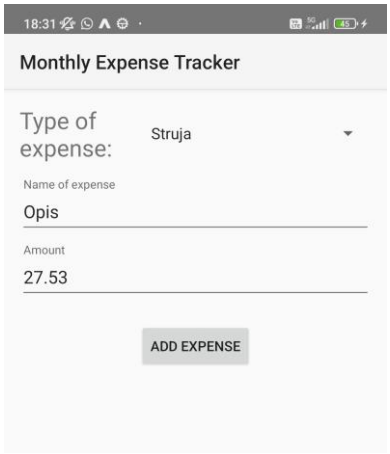


Sl. 5.6. Skeniranje barkoda

Za testiranje skeniranja računa koristi se generator naloga za nacionalna plaćanja [11] koji generira račun s PDF417 linijskim kodom nakon popunjavanja određenih stavki (Slika 5.7.), kojeg se preko Android uređaja skenira na monitoru. Nakon unesenih svih potrebnih stavki, generator prikaže PDF417 linijski kod koji sadrži unesene podatke.

Sl. 5.7. Račun za testiranje

Nakon što aplikacija uspješno skenira linijski kod otvori se aktivnost za dodavanje troška gdje se vide naziv i iznos računa/troška koji su automatski popunjeni podacima s generiranog naloga (Slika 5.8.).



Sl. 5.8. Popunjena polja nakon skeniranja računa

Pritiskom na padajući izbornik moguće je odabrati tip troška (Slika 5.9.), u padajućem izborniku prikazuju se tipovi troškova koje je korisnik već unio. Ukoliko ni jedan od postojećih tipova nije tip troška koji korisnik želi dodati, postoji mogućnost dodavanja novog tipa troška koji će se kasnije opet moći koristiti aktivacijom dugmeta „Add new type“.

Sl. 5.9. Izgled padajućeg izbornika za tip troška

Aktivacijom dugmeta „Add a new type“ korisniku se prikaže dijalog za unos novog tipa troška (Slika 5.10.). Nakon potvrde novi tip troška se spremi u padajući izbornik čiju trenutnu vrijednost, zajedno s imenom troška, iznosom i datumom spremaju na bazu.

Sl. 5.10. Otvoren dijalog nakon stiska tipke „Add new type“

6. ZAKLJUČAK

U ovom završnom radu razvijena je Android aplikacija za praćenje režijskih troškova kućanstva. U usporedbi s ostalim aplikacijama na tržištu ova aplikacija nema toliko funkcionalnosti kao najpopularnije aplikacije, poput integracije s bankovnim računom ili karticom. Jedna od funkcionalnosti izrađene aplikacije koju nemaju druge aplikacije na tržištu je skeniranje hrvatskog naloga za nacionalna plaćanja. Aplikacija bi se mogla primijeniti za jednostavno praćenje režijskih i ostalih troškova unutar kućanstva. Osnovne funkcionalnosti tražene opisom zadatka završnog rada su zadovoljene, poput spremanja, dohvaćanja i prikaza podataka o troškovima, te skeniranje naloga za nacionalna plaćanja, ali ima prostora za napredak i nadogradnju aplikacije kako bi imala više funkcionalnosti za statističku analizu troškova koje bi znatno povećale korisnost aplikacije.

LITERATURA

[1] Firebase Firestore dokumentacija, dostupno na: https://firebase.google.com [rujan, 2023.]

[2] Wikipedia, PDF417 linijski kod, dostupno na: https://en.wikipedia.org/wiki/PDF417 [rujan, 2023.]

[3] Google ML kit upute, dostupno na: https://developers.google.com/ml-kit/guides [rujan, 2023.]

[4] MPAndroidChart repozitorij, dostupno na: https://github.com/PhilJay/MPAndroidChart [rujan, 2023.]

[5] Mint Android aplikacija, dostupno na: https://mint.intuit.com/ [rujan, 2023.]

[6] Wallet Android aplikacija, dostupno na: https://budgetbakers.com/ [rujan, 2023]

[7] Spendee Android aplikacija, dostupno na: https://www.spendee.com/ [rujan, 2023]

[8] Kotlin programski jezik, dostupno na: https://kotlinlang.org/ [rujan, 2023]

[9] D., Jemerov, S., Isakova, Kotlin in action, Manning Publications, Sjedinjene Američke Države, 2017.

[10] Android studio, dostupno na: https://developer.Android.com/studio [rujan, 2023]

[11] Generator naloga za nacionalna plaćanja, dostupno na: https://knee-cola.github.io/generator-opce-uplatnice/ [rujan, 2023]

SAŽETAK

U ovom završnom radu razvijena je mobilna Android aplikacija koja omogućava praćenje režijskih troškova kućanstva. Korisnici mogu ručno ili automatski skeniranjem računa pomoću kamere dodati određeni trošak koji ima naziv, tip i iznos. Korisnik može svoje troškove pratiti pomoću kružnog dijagrama na početnoj stranici aplikacije ili u obliku liste. Aplikacija je izgrađena u Kotlin programskom jeziku unutar programskog okruženja Android Studio. U radu su objašnjene ključne funkcionalnosti aplikacije popraćene slikama programskog koda.

Ključne riječi: Android, aplikacija, Kotlin, trošak

ABSTRACT

Mobile application for recording household running costs

In this final paper, a mobile Android application that enables the monitoring of household running costs was developed. Users can add a specific expense that has a name, type, and amount either manually or automatically by scanning a receipt through the camera. The user can then track his expenses through a pie chart on the home page of the application or in the form of a list. The application was built in the Kotlin programming language within the Android Studio programming environment. The paper explains the key functionalities of the application accompanied by images of the program code.

Keywords: Android, application, cost, Kotlin