Aplikacija za praćenje troškova kućanstva

Kuridža, Igor

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:750779

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-17**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni preddiplomski studij računarstva

APLIKACIJA ZA PRAĆENJE TROŠKOVA KUĆANSTVA

Završni rad

Igor Kuridža

Osijek, 2019.

Sadržaj

1. UVOD 1
1.1. Zadatak završnog rada 1
2. OPIS KORIŠTENIH TEHNOLOGIJA 2
2.1. Android Studio 2
2.1.1. Manifest 2
2.1.2. *Gradle build* skripta 3
2.1.3. Resursi 4
2.2. Programski jezik Kotlin 4
2.3. XML 5
2.4. *Activity* 5
2.5. Fragment 6
2.6. RecyclerView 7
2.6.1. *LayoutManager* 7
2.6.2. *ViewHolder* 7
2.6.3. *Adapter* 8
2.7. *ViewPager* 8
2.8. *Room* baza podataka 9
2.9. *Shared Preferences* 10
2.10. Koin 11
2.11. Model-Pogled-Prezenter (MVP) arhitekturni obrazac 12
2.12. Korištene biblioteke potrebne za razvoj aplikacije 13
3. ANDROID APLIKACIJA ZA PRAĆENJE TROŠKOVA KUĆANSTVA 15
3.1. Pokretanje aplikacije 15
3.2. Unos troškova i primanja 15
3.3. Transakcije 16
3.4. Računi 17

3.5. Pregled – prikaz transakcija na grafu 18
3.6. Ravnoteža primanja i troškova 18
4. IZGLED I KORIŠTENJE APLIKACIJE 20
4.1. Bez unesenih podataka 20
4.2. Unošenje podataka 21
4.3. S unesenim podatcima 22
4.3.1. Kategorije troškova i primanja 23
4.3.2. Računi 23
4.3.3. Transakcije 24
4.3.4. Pregled transakcija na grafu 25
4.3.5. Ravnoteža 26
5. ZAKLJUČAK 28
LITERATURA 29
SAŽETAK 30
ABSTRACT 31
ŽIVOTOPIS 32
PRILOZI 33

1. UVOD

Kroz mjesec dana jedno kućanstvo ima dosta primanja i troškova koje je potrebno pratiti kako bi se održavala pozitivna bilanca. Prije su se sva primanja i troškovi stavljali na papir i ručno se izračunavalo koliko se dobiti ili gubitka ostvarilo za određeno razdoblje. Napretkom tehnologije danas su dostupni različiti programi i aplikacije koje nam omogućuju laku evidenciju novčanog toka. Podjela troškova i primanja po kategorijama, lako računanje ukupnih osobnih primanja i troškova, grafički prikaz istih za željeno vremensko razdoblje, su samo neke mogućnosti od modernih aplikacija za praćenje novčanog toka.

U ovom radu opisana je jedna takva aplikacija koja omogućuje korisniku praćenje troškova i primanja, ali unutar kućanstva. Tema rada razrađena je u 5 poglavlja. Prvo poglavlje je sami uvod u kojem je ukratko opisano koja tema će biti obrađena u radu. U drugom poglavlju teorijski su opisane korištene tehnologije. U poglavlju tri dana je realizacija aplikacije u „Android Studio“ integriranom razvojnom okruženju za razvoj aplikacija za Android, te je opisan programski kod. U četvrtom poglavlju se prikazuje izgled aplikacije te njezino korištenje. U petom poglavlju se iznosi zaključak.

1.1. Zadatak završnog rada

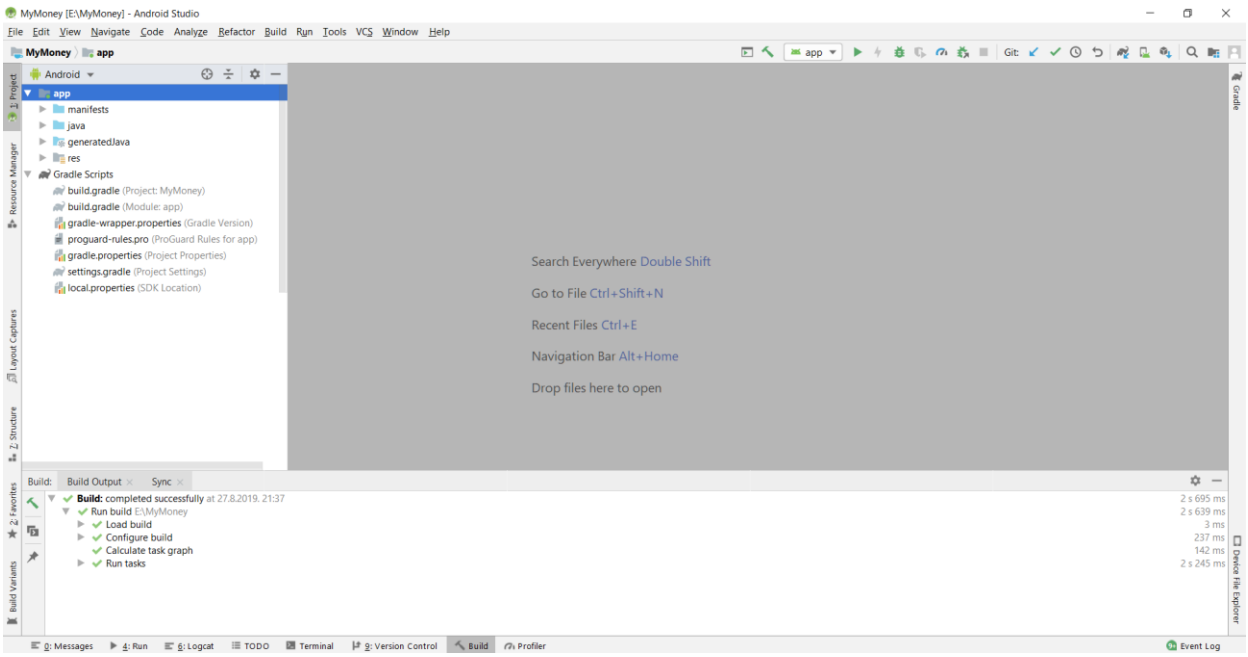
Aplikacija treba omogućiti korisnicima unos podataka o primanju jednog kućanstva, troškovima, računima, režijama te bilo kakvim drugim novčanim transakcijama. Aplikacija treba na grafički način prikazivati prihode i rashode na mjesečnoj i godišnjoj razini s obzirom na razne kategorije prihoda i troškova.

2. OPIS KORIŠTENIH TEHNOLOGIJA

U ovom djelu će biti opisan Android Studio te njegove najbitnije datoteke. Osim toga ukratko će biti opisan programski jezik Kotlin te će se navesti njegove prednosti. Spomenut će se i osnove XML-a (engl. *eXtensible Markup Language*), proširivog opisnog jezika. Bit će opisane korištene komponente poput *RecyclerViewa, ViewPagera, Room* baze podataka, *Koina*, aktivnosti (engl. *activity*) i fragmenata. Također će se opisati primijenjeni arhitekturni obrazac Model-Pogled-Prezenter (engl. *Model-View-Presenter*, MVP). Na kraju će se navesti korištene biblioteke potrebne za razvoj aplikacije.

2.1. Android Studio

Android Studio je službeno integrirano razvojno okruženje (engl. *Integrated Development Environment,* IDE) za Google-ov operacijski sustav Android. Razvijen je za Android kako bi ubrzao razvoj i pomogao u izradi najkvalitetnijih aplikacija za svaki Android uređaj. Može se preuzeti za preferiranu platformu Windows, Mac OS X ili Linux. Android Studio uključuje mogućnost testiranja aplikacije na stvarnom uređaju ili na emulatoru [1]. Na slici 2.1. je prikazano sučelje Android Studia.



Slika 2.1. Sučelje Android Studio integriranog razvojnog okruženja

2.1.1. Manifest

Jedna od temeljnih datoteka unutar Android aplikacije je Manifest datoteka, koja se nalazi unutar *app-manifests* mape u strukturi projekta. To je datoteka pisana XML opisnim jezikom koja

sadrži meta podatke o aplikaciji, definira strukturu aplikacije, sve aktivnosti (engl. *activities*), servise, dozvole, definira ikonu, verziju aplikacije i slično. Svaki gradivni element aplikacije mora biti prijavljen i definiran u Manifest datoteci [2]. U ovoj aplikaciji postoji samo jedna aktivnost (engl. *activity*) nazvana „*MainActivity*“ te je uporabom filtera namjere (engl. *intent filter*) navedeno da se ta aktivnost pokreće prilikom pokretanja aplikacije. Ako je aplikacijska klasa (engl. *application class*) posebno kreirana, onda je potrebno pod oznakom *name* u Manifest datoteci navesti putanju koja pokazuje na aplikacijsku klasu, kako bi se osiguralo da se koristi ta posebno kreirana klasa. Manifest datoteka ove aplikacije je prikazana slikom 2.2.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="igorkuridza.ferit.hr.mymoney">

    <application
            android:name=".MyMoneyApp"
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="MyMoney"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
            <activity android:name=".ui.activities.MainActivity"
                      android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Slika 2.2 „*AndroidManifest.xml*“

2.1.2. *Gradle build* skripta

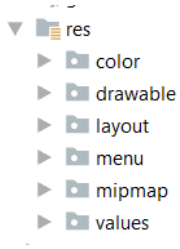
U *gradle build* skripti (Slika 2.3.) navode se aplikacijski ID, trenutna inačica aplikacije, minimalna SDK (komplet za razvoj softvera, engl. *Software Development Kit*) verzija, ciljana SDK verzija i slično. Aplikacijski ID jedinstveno identificira aplikaciju na uređaju i u *Google Play* trgovini. Minimalna SDK verzija predstavlja minimalnu verziju operacijskog sustava Android koja je potrebna za pokretanje aplikacije. Ciljana SDK verzija predstavlja verziju operacijskog sustava Android za koju je kreirana aplikacija.

```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "igorkuridza.ferit.hr.mymoney"
        minSdkVersion 19
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

Slika 2.3. Dio *gradle build* skripte

2.1.3. Resursi

Resursi (Slika 2.4.) predstavljaju slike, izgled aplikacije, nizove (engl. *strings*) i slično. Oni se prilikom razvoja aplikacije za Android platformu razdvajaju od koda te smještaju u *res* mapu unutar projekta. To razdvajanje olakšava izmjene i prilagodbe različitim zahtjevima, jer je lako imati više različitih inačica istog resursa [3]. Naprimjer, razdvajanje nizova u *strings.xml* omogućuje jednostavno prevođenje aplikacije na više jezika. Izgled Android aplikacije se definira resursima koji se nazivaju izgled (engl. *layout*).



Slika 2.4. Resursi

2.2. Programski jezik Kotlin

Kotlin je programski jezik izvrstan za razvoj Android aplikacija. Donosi sve prednosti modernog jezika na Android platformu bez uvođenja novih ograničenja. Nekoliko je stvari zbog kojih Kotlin izvrsno odgovara Androidu:

- Kompatibilnost: Kompatibilan je sa JDK (Java razvojni komplet, engl. *Java Development Kit*) 6, osiguravajući da se Kotlin aplikacije bez problema mogu pokretati na starijim Android uređajima.
- Izvođenje: Kotlin aplikacije se izvode isto brzo kao i jednaka Java aplikacija.
- Interoperabilnost: Java i Kotlin su u potpunosti interoperabilni. To znači da se Kotlin kod može pokretati u Java datotekama i obratno.
- Učenje: Kotlin je jednostavan za naučiti, posebno za ljude koji su navikli na moderne jezike. Pretvarač Jave u Kotlin u *IntelliJ* integriranom okruženju za razvoj softvera i Android Studio-u čini ga još lakšim za učenje. Java programski kod i Kotlin programski kod mogu se koristiti u istom projektu [4].

Kotlin je moderan jezik koji nam s obzirom na Javu omogućuje da istu funkcionalnost napišemo s puno manje koda, te je ujedno i pregledniji nego Java programski kod.

2.3. XML

XML je opisni jezik pomoću kojeg definiramo dizajn aplikacije. Pomoću njega definiramo izgled aktivnosti, fragmenta i slično. Na slici 2.5. je vidljivo kako se XML piše. Unutar XML opisnog jezika definiraju se elementi korisničkog sučelja. Unutar aplikacije se ti elementi „napuhuju“ (engl. *inflate*) te se na temelju njih stvaraju objekti poput polja za prikaz podataka, polja za unos i slično.

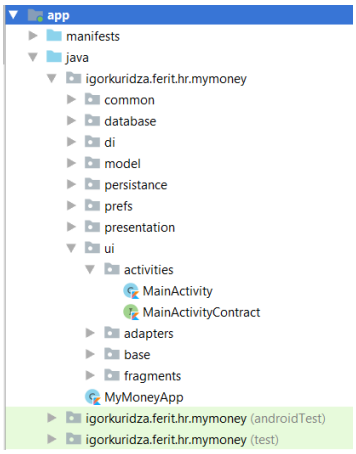
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.activities.MainActivity">

    <FrameLayout
            android:id="@+id/fragmentContainer"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toTopOf="@id/bottomNavigation"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent">
    </FrameLayout>

    <com.google.android.material.bottomnavigation.BottomNavigationView
            android:id="@+id/bottomNavigation"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:itemIconTint="@color/bottom_nav_colors"
            app:itemTextColor="@color/bottom_nav_colors"
            app:itemIconSize="30dp"
            app:menu="@menu/bottom_nav"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            android:background="?android:attr/windowBackground"
            />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 2.5. XML za *MainActivity*

2.4. *Activity*

Activity je klasa koja predstavlja jedan zaslon aplikacije. Za razliku od klasičnih desktop aplikacija, Android aplikacije nemaju glavnu (engl. *main*) metodu koja služi kao ulazna točka programa, već se životni ciklus pojedinih aktivnosti oslanja na određene metode povratnog poziva (engl. *callback*). Te metode se primjerice pozivaju kod stvaranja aktivnosti ili kod gašenja, a metode životnog ciklusa su: *onCreate*(), *onStart*(), *onResume*(), *onPause*(), *onStop*(), *onRestart*() i *onDestroy*() [5].

U ovoj aplikaciji postoji jedan *Activity* i nalazi se u paketu (engl. *package*) *ui-activities* (Slika 2.6.).



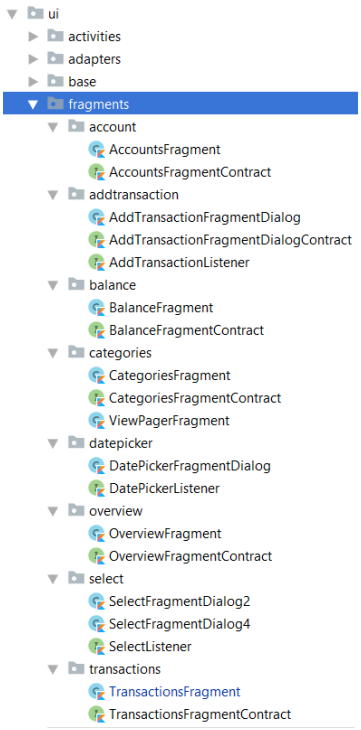
Slika 2.6. *MainActivity* unutar paketa

2.5. Fragment

Fragment je dodatan sloj između aktivnosti i sučelja. Predstavlja klasu koja omogućuje modularan dizajn aktivnosti. Možemo kombinirati više fragmenata unutar jedne aktivnosti, odnosno jedna aktivnost može sadržavati više fragmenata. O fragmentu se može razmišljati kao o modularnom odjeljku aktivnosti koji ima vlastiti životni ciklus koji je usko vezan za aktivnost u kojoj se nalazi. Prima vlastite ulazne događaje koje možemo dodavati ili uklanjati dok se aktivnost izvodi [6].

U ovoj aplikaciji postoji šest fragmenata te četiri fragment dijaloga koji se nalaze u paketu *ui-fragments* (Slika 2.7.).

Fragment dijalog je fragment koji prikazuje dijaloški prozor. Sadrži dijaloški objekt koji se prema potrebi prikazuje na temelju stanja fragmenta.



Slika 2.7. Fragmenti unutar paketa

2.6. RecyclerView

RecyclerView omogućuje prikazivanja velikog broja podataka u obliku liste koja se može pomicati gore ili dolje. Kako bi *RecyclerView* funkcionirao mora imati prilagodnika podataka (engl. *adapter*), držača pogleda (engl. *view holder*) te upravljača rasporeda pogleda (engl. *layout manager*). Velika prednost *RecyclerView-a* je korištenje upravljača rasporeda pogleda [7].

2.6.1. *LayoutManager*

LayoutManager utječe na raspored i ponašanje elemenata liste u *RecyclerView-u*. Najosnovniji *LayoutManager-i* su *LinearLayoutManager* (prikazuje elemente u obliku liste jedan ispod drugoga), *GridLayoutManager* (prikazuje elemente u obliku dvodimenzionalne rešetke koja se može pomicati gore ili dolje, a elementi su iste visine i širine) i *StaggeredGridLayoutManager* (isto kao *GridLayoutManager*, ali elementi mogu biti različite visine i širine).

2.6.2. *ViewHolder*

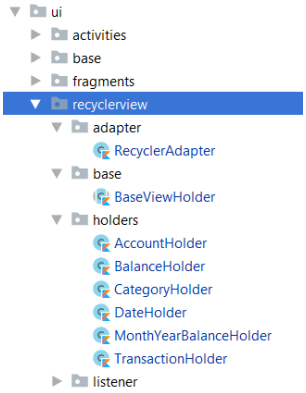
ViewHolder je klasa koja predstavlja jedan element *RecyclerView*-a. *RecyclerView* će napraviti onoliko instanci *ViewHolder*-a koliko je potrebno da se popuni jedan ekran podatcima. Za sve ostale podatke koji se ne nalaze na ekranu reciklirat će se postojeći *ViewHolder* te popuniti

novim podatcima. U ovoj aplikaciji *BaseViewHolder* se nalazi unutar paketa *ui-recyclerview-base*, a ostali *ViewHolder*-i se nalaze unutar *ui-recyclerview-holders* (Slika 2.8.).

2.6.3. *Adapter*

Adapter je klasa koja služi za držanje podataka koji će se prikazivati na zaslonu i za kreiranje *ViewHolder*-a. *Adapter* sadrži tri sistemske metode: *getItemCount*()- metoda koja vraća ukupni broj elemenata koji će se prikazivati, *onCreateViewHodler*()- metoda koja služi za kreiranje i recikliranje *ViewHolder*-a i *onBindViewHolder*()- metoda koja povezuje podatke unutar adaptera sa *ViewHolder*-om. U ovoj aplikaciji adapter se nalazi unutar paketa *ui-recyclerview-adapter* (Slika 2.8.).

U aplikaciji se koristio *BaseViewHolder* kako bi se mogao koristiti samo jedan *RecyclerView* adapter. *RecyclerAdapter* ima još jednu dodatnu metodu *getItemViewType()* – metoda koja se koristi kada se žele prikazivati različiti tipovi podataka.

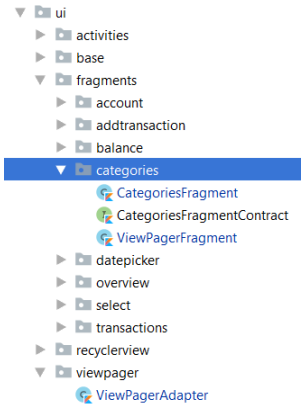


Slika 2.8. Holderi i adapter unutar paketa

2.7. *ViewPager*

ViewPager je komponenta koja se koristi za prikazivanje fragmenata te omogućava korisniku da prelazi lijevo ili desno za prikazivanje drugog fragmenta kliznim pomakom po ekranu [8]. Kako bi se *ViewPager* mogao popunjavati fragmentima potrebno je napisati adapter za *ViewPager*. *ViewPagerAdapter* predstavlja klasu koja je odgovorna za popunjavanje fragmenata unutar *ViewPager*-a.

U ovoj aplikaciji *ViewPagerFragment* se nalazi u paketu *ui-fragments-categories*, a *ViewPagerAdapter* u paketu *ui-viewpager* (Slika 2.9.).



Slika 2.9. *ViewPagerFragment* i *ViewPagerAdapter* unutar paketa

2.8. *Room* baza podataka

Room baza podataka pruža sloj apstrakcije preko strukturiranog jezika upita (engl. *Structured Query Language*, SQL) kako bi se omogućio pristup bazi podataka uz istovremeno korištenje pune snage *SQLite*-a. *Room* stvara predmemoriju podataka aplikacije na uređaju koji pokreće aplikaciju. Ova predmemorija, koja služi kao jedinstveni izvor podataka aplikacije, omogućava korisnicima da pregledavaju podatke unutar aplikacije, bez obzira na to ima li internetske veze. Postoje tri glavne komponente: objekt pristupa podatcima (engl. *Data Access Object*, DAO), entitet i baza podataka (engl. *database*). Entitet (Slika 2.10.) predstavlja podatke za jedan redak tablice, izrađen pomoću označenog Java podataka objekta (engl. *Plained old Java object*, POJO). DAO (Slika 2.11.) definira metode koje pristupaju bazi podataka, koristeći napomenu za vezanje SQL-a za svaku metodu. *Database* (Slika 2.12.) je klasa nositelja koja koristi napomene za definiranje popisa entiteta i verzije baze podataka. Sadržaj te klase definira popis objekata pristupa podatcima, tj. popis DAO-va [9].

```
@Entity(tableName = "account")
data class Account(
    @PrimaryKey(autoGenerate = true)
    val id: Long? = null,
    @ColumnInfo(name = "accountName")
    val name: String,
    @ColumnInfo(name = "accountAmountOfMoney")
    val amountOfMoney: Float,
    val image: Int,
    @ColumnInfo(name = "accountColor")
    val color: Int
)
```

Slika 2.10. Entitet *Account*

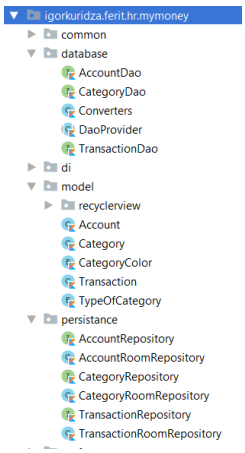
```
@Dao
interface AccountDao {

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun addAccount(account: Account)

    @Query( value: "SELECT *FROM account")
    fun getAllAccounts(): List<Account>

    @Query( value: "UPDATE account SET accountAmountOfMoney= :amountOfMoney WHERE id= :id")
    fun updateAccountAmountOfMoney(amountOfMoney: Float, id: Long)

    @Query( value: "SELECT * FROM account WHERE accountName= :name")
    fun getAccountByName(name: String): Account
}
```

Slika 2.11. *AccountDao*

```
@Database(entities = [Transaction::class, Category::class, Account::class], version = 1, exportSchema = false)
@TypeConverters(Converters::class)
abstract class DaoProvider: RoomDatabase() {

    abstract fun transactionDao(): TransactionDao
    abstract fun categoryDao(): CategoryDao
    abstract fun accountDao(): AccountDao

    companion object {
        private var instance: DaoProvider? = null

        fun getInstance(context: Context): DaoProvider{
            if(instance==null){
                instance = Room.databaseBuilder(
                    context.applicationContext,
                    DaoProvider::class.java,
                    name: "TransactionDB"
                ).fallbackToDestructiveMigration()
                    .allowMainThreadQueries()
                    .build()
            }
            return instance as DaoProvider
        }
    }
}
```

Slika 2.12. *DaoProvider*

U aplikaciji je implementirano više DAO sučelja (engl. *interface*) i entiteta jer postoji više podataka koje je potrebno spremiti u bazu podataka. Dodane su *RoomRepository* klase koje služe za interakciju s *Room* bazom podataka i u njima su implementirane metode koje koristi DAO. Neke od metoda su metode za dodavanje podataka u bazu, metode za dohvaćanje podataka i slično. U ovoj aplikaciji DAO i *Database* se nalaze u paketu *database*, entiteti se nalaze u paketu *model* i *Room* repozitoriji se nalaze u *persistance* paketu, kao što je prikazano na slici 2.13.



Slika 2.13. Komponente *Room* baze podataka unutar paketa

2.9. *Shared Preferences*

Shared Preferences sučelje se koristi ako je potrebno spremiti podatke u obliku ključ-vrijednost. Objekt *Shared Preferences* upućuje na datoteku koja sadrži parove ključ-vrijednost i

pruža jednostavne metode za njihovo čitanje i spremanje. Svaka datoteka *Shared Preferences* može biti privatna i dijeljena. [10]

U ovoj aplikaciji je korišteno sučelje *Shared Preferences* (Slika 2.14.) kako bi se spremio logički (engl. *boolean*) tip podatka koji predstavlja prvo pokretanje aplikacije.

```
class SharedPrefsHelperImpl(): SharedPrefsHelper {

    private val preferences = MyMoneyApp.instance.providePreferences()

    override fun storeFirstStartOfApp(firstStartOfApp: Boolean) {
        preferences.edit().putBoolean(PREFS_KEY,firstStartOfApp).apply()
    }

    override fun getFirstStartOfApp(): Boolean {
        return preferences.getBoolean(PREFS_KEY, true)
    }
}

fun provideSharedPrefs(): SharedPrefsHelper {
    return SharedPrefsHelperImpl()
}
```

Slika 2.14. *SharedPrefsHelperImpl*

2.10. Koin

Koin se koristi za ubrizgavanje ovisnosti (engl. d*epedency injection*). Koristi Kotlinove DSL-ove (jezik specifičan za domenu, engl. *Domain Specific Language*) za rješavanje grafa ovisnosti za vrijeme izvođenja [11].

Ubrizgavanje ovisnosti je tehnika kojom jedan objekt (ili statička metoda) opskrbljuje ovisnost drugog objekta. Ovisnost je objekt koji se može koristiti. Odgovornost ubrizgavanja ovisnosti je stvaranje objekata, znati koje klase zahtijevaju te objekte te im pružiti te objekte [12].

Prvo se opisuju ovisnosti u Koin modulima (Slika 2.15. i slika 2.16.). Nakon opisivanja modula poziva se *startKoin* funkcija kojoj se predaje *Android Context* i lista modula (Slika 2.17.).

```
val presentationModule = module { this: Module

    factory<AccountsFragmentContract.Presenter>{ AccountsFragmentPresenter(get()) }
    factory<AddTransactionFragmentDialogContract.Presenter> { AddTransactionFragmentDialogPresenter(get(), get()) }
    factory<BalanceFragmentContract.Presenter> { BalanceFragmentPresenter(get(), get())}
    factory<CategoriesFragmentContract.Presenter> {CategoriesFragmentPresenter(get(), get())}
    factory<MainActivityContract.Presenter> {MainActivityPresenter(get(), get(), get())}
    factory<OverviewFragmentContract.Presenter>{ OverviewFragmentPresenter(get(), get()) }
    factory<TransactionsFragmentContract.Presenter> { TransactionsFragmentPresenter(get()) }
}
```

Slika 2.15. *presentationModule*

```
val repositoryModule = module { this: Module

    factory<AccountRepository> { AccountRoomRepository() }
    factory<TransactionRepository>{ TransactionRoomRepository()}
    factory<CategoryRepository> { CategoryRoomRepository() }
    factory { provideSharedPrefs() }
}
```

Slika 2.16. *repositoryModule*

```
startKoin { this: KoinApplication
    androidContext( androidContext: this@MyMoneyApp)
    modules(listOf(presentationModule, repositoryModule))
}
```

Slika 2.17. Funkcija *startKoin* unutar *MyMoneyApp* klase

2.11. Model-Pogled-Prezenter (MVP) arhitekturni obrazac

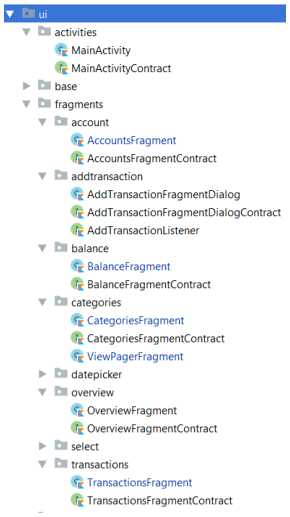
MVP (engl. *Model-View-Presenter*) obrazac omogućava razdvajanje aplikacije na tri sloja: model sloj, prezentacijski sloj i sloj pogleda (engl. *view*). Svaki sloj je definiran i međusobno su povezani preko slabe reference (engl. *weak reference*), najčešće preko sučelja.

Model sloj predstavlja opskrbljivača podataka koje želimo prikazati na ekranu. Komunicira s uslugama poput baze podataka i slično.

Sloj pogleda služi za prikaz sučelja te podataka na sučelju. Njegova zadaća je javiti prezentacijskom sloju vrstu korisničke interakcije, odnosno zahtjev korisnika.

Prezentacijski sloj predstavlja „čovjeka u sredini“, između sloja pogleda i model sloja. Njegova zadaća je povezati korisnikovu interakciju (poput pritiska na ekran ili pisanja na tipkovnici) sa model slojem. Model sloj dohvaća potrebne podatke, obrađuje ih i vraća prezentacijskom sloju koji prosljeđuje obrađene podatke sloju pogleda.

U ovoj aplikaciji slojevi su međusobno povezani preko Kotlin sučelja nazvanih ugovori (engl. *contracts*), (Slika 2.18.). Oni predstavljaju ugovore između sloja pogleda i prezentacijskog sloja što se može vidjeti na slici 2.19.

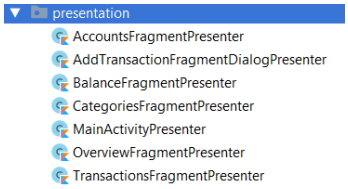


Slika 2.18. Ugovori

```
interface AccountsFragmentContract {

    interface View{

        fun onGetAccounts(accounts: List<Account>)

        fun onGetTotalAccountAmount(totalAccountAmount: Float)
    }

    interface Presenter: BasePresenter<View>{

        fun getAccounts()

        fun getTotalAccountAmount()
    }
}
```

Slika 2.19. *AccountFragmentContract*

U ovoj aplikaciji prezenteri predstavljaju prezentacijski sloj i nalaze se u paketu *presentation* kao što je vidljivo na slici 2.20.



Slika 2.20. Prezenteri

2.12. Korištene biblioteke potrebne za razvoj aplikacije

Korištene biblioteke su *Room* biblioteka (omogućava korištenje *Room* baze podataka i svih njenih mogućnosti), biblioteka za dizajn, *RecyclerView* biblioteka (potrebna kako bi se mogao koristiti *RecyclerView*), *MPAndroidChart* biblioteka za prikazivanje grafova te Koin biblioteka za korištenje Koina. Implementacija korištenih biblioteka prikazana je na slici 2.21.

```
//room
implementation "androidx.room:room-runtime:$room_version"
kapt "androidx.room:room-compiler:$room_version"

//materialDesign
implementation 'com.google.android.material:material:1.0.0'

//recyclerView
implementation 'androidx.recyclerview:recyclerview:1.0.0'

//graphView
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

//koin
implementation "org.koin:koin-android:2.0.1"
```

Slika 2.21. Korištene biblioteke

3. ANDROID APLIKACIJA ZA PRAĆENJE TROŠKOVA KUĆANSTVA

U ovom djelu će biti opisan i prikazan postupak izrade aplikacije te detaljnije objašnjena implementacija korištenih tehnologija.

3.1. Pokretanje aplikacije

Prilikom pokretanja aplikacije pokreće se *MainActivity* sa navigacijom na dnu (engl. *bottom navigation*) i fragment sa kategorijama troškova. Ako se aplikacija prvi put pokrenula poziva se metoda u prezenteru za spremanje računa te kategorija troškova i primanja u *Room* bazu podataka (Slika 3.1.). Na slici 3.2. je vidljiva funkcija za spremanje potrebnih podataka u *Room* bazu podataka.

```
override fun onGetFirstStartOfApp(firstStartOfApp: Boolean) {
    if(firstStartOfApp) {
        presenter.apply { this: MainActivityContract.Presenter
            saveCategoriesAndAccountsToDb()
            saveFirstStartOfApp( firstStartOfApp: false)
        }
    }
}
```

Slika 3.1. Funkcija u *MainActivityPresenter* za spremanje računa te kategorija troškova i primanja

```
override fun saveCategoriesAndAccountsToDb() {
    categoryRepository.apply { this: CategoryRepository
        addCategory(Category(name = "Hrana i piće", color = CategoryColor.FIRST, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Režije stanovanja", color = CategoryColor.SECOND, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Prijevoz", color = CategoryColor.THIRD, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Odjeća i obuća", color = CategoryColor.FOURTH, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Obrazovanje", color = CategoryColor.FIFTH, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Zabava", color = CategoryColor.SIXTH, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Putovanja", color = CategoryColor.SEVENTH, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Zdravlje", color = CategoryColor.EIGHT, typeOfCategory = TypeOfCategory.EXPENSES))
        addCategory(Category(name = "Ostalo", color = CategoryColor.NINTH, typeOfCategory = TypeOfCategory.EXPENSES))

        addCategory(Category(name = "Plaća", color = CategoryColor.NINTH, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Nagrade i bonusi", color = CategoryColor.EIGHT, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Dohodak od imovine", color = CategoryColor.SEVENTH, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Dohodak od nesamostalnog rada", color = CategoryColor.SIXTH, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Dohodak od samostalne djelatnosti", color = CategoryColor.FIFTH, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Mirovine", color = CategoryColor.FOURTH, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Naknade za nezaposlenost", color = CategoryColor.THIRD, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Primljeni pokloni", color = CategoryColor.SECOND, typeOfCategory = TypeOfCategory.INCOMES))
        addCategory(Category(name = "Ostala tekuća primanja", color = CategoryColor.FIRST, typeOfCategory = TypeOfCategory.INCOMES))
    }

    accountRepository.apply { this: AccountRepository
        addAccount(Account(name = "Kartica", image = R.mipmap.ic_account_credit_card, amountOfMoney = 0F, color = R.color.accountCardColor))
        addAccount(Account(name = "Gotovina", image = R.mipmap.ic_account_cash, amountOfMoney = 0F, color = R.color.accountCashColor))
    }
}
```

Slika 3.2. Funkcija u *MainActivityPresenter* za spremanje potrebnih podataka u *Room* bazu podataka

3.2. Unos troškova i primanja

Prilikom pritiska na jednu od kategorija troškova ili primanja (Slika 3.3.) otvara se *AddTransactionFragmentDialog*, dijalog za dodavanje transakcija, koji s obzirom na tip kategorije

omogućuje unos količine novca, bilješke te odabir računa. Pod bilješku se može unijeti npr. potkategorija kako bi se detaljnije specificiralo za šta novac odlazi ili dolazi.

```
override fun onItemClick(item: Any) {
    val fragmentManager = fragmentManager
    val addTransactionFragmentDialog = AddTransactionFragmentDialog.newInstance(item as Category, tvDate.text.toString())
    addTransactionFragmentDialog.setAddTransactionListener(this)
    addTransactionFragmentDialog.show(fragmentManager,  tag: "")
}
```

Slika 3.3. Funkcija koja se pokreće kada se kliknulo na neku od kategorija

Kada se unesu potrebni podatci i pritisne gumb „Spremi“, podatci se spremaju u *Room* bazu podataka (Slika 3.4.), a količina unesenog novca se oduzima od ukupne količine novca koju sadrži odabrani račun ako se radi o troškovima, a zbraja ako se radi o primanjima (Slika 3.5.).

```
private fun saveTransaction(account: Account){
    val amountOfMoney = getAmountOfTransactionFromEditText()
    if(isAmountOfMoneyEmptyOr0(amountOfMoney)){
        dismiss()
    }
    else {
        val note = getNoteFromEditText()
        presenter.updateAccountAmount(account, amountOfMoney, isTypeOfCategoryExpenses())
        val transaction = convertToTransaction(category,date,amountOfMoney,account, note)
        presenter.addTransactionToDb(transaction)
    }
}
```

Slika 3.4. Funkcija za spremanje transakcije

```
override fun addTransactionToDb(transaction: Transaction) {
    transactionRepository.addTransaction(transaction)
}

override fun updateAccountAmount(account: Account, amount: Float, isTypeOfCategoryExpense: Boolean) {
    if(isTypeOfCategoryExpense) accountRepository.updateAccountAmountOfMoney( amountOfMoney: account.amountOfMoney - amount, account.id!!)
    else accountRepository.updateAccountAmountOfMoney( amountOfMoney: account.amountOfMoney + amount, account.id!!)
}
```

Slika 3.5. Funkcije u *AddTransactionsPresenter*

3.3. Transakcije

Transakcije se prikazuju u fragmentu *TransactionsFragment*. S obzirom na odabir korisnika omogućen je prikaz transakcija za cijelo vrijeme, današnji datum, trenutni mjesec i trenutnu godinu (Slika 3.6.).

```
override fun getTransactionAdapterData(type: String) {
    when(type){
        TYPE_ALL_TIME -> {
            val transactions = getAllTransactions()
            val newDataForAdapter = getNewDataForAdapter(transactions)
            view.onGetTransactionAdapterData(newDataForAdapter)
        }
        TYPE_TODAY ->{
            val transactions = getTransactionsFromToday()
            val newDataForAdapter = getNewDataForAdapter(transactions)
            view.onGetTransactionAdapterData(newDataForAdapter)
        }
        TYPE_CURRENT_MONTH ->{
            val currentMonth = getCurrentMonth()
            val transactions = getTransactionsByMonthYear(currentMonth)
            val newDataForAdapter = getNewDataForAdapter(transactions)
            view.onGetTransactionAdapterData(newDataForAdapter)
        }
        TYPE_CURRENT_YEAR ->{
            val currentYear = getCurrentYear().toInt()
            val transactions = getTransactionsByYear(currentYear)
            val newDataForAdapter = getNewDataForAdapter(transactions)
            view.onGetTransactionAdapterData(newDataForAdapter)
        }
    }
}
```

Slika 3.6. Funkcija u *TransactionsFragmentPresenter* za dohvaćanje podataka za adapter

Nakon odabranog željenog vremena prikaza transakcija od strane korisnika, transakcije se dohvaćaju iz *Room* baze podataka i se prikazuju na zaslonu (Slika 3.7.).

```
private fun getTransactionsFromToday(): List<Transaction>{
    val todayDate = getTodayDate()
    return transactionsRepository.getAllTransactionsByDate(todayDate)
}

private fun getAllTransactions(): List<Transaction>{
    return transactionsRepository.getAllTransactions()
}

private fun getTransactionsByYear(year: Int): List<Transaction>{
    return transactionsRepository.getTransactionsByYear(year)
}

private fun getTransactionsByMonthYear(monthYear: String): List<Transaction>{
    return transactionsRepository.getTransactionsByMonthYear(monthYear)
}
```

Slika 3.7. Funkcije u *TransactionsFragmentPresenter* za dohvaćanje transakcija

3.4. Računi

Računi se nalaze u fragmentu *AccountsFragment*. Iz *Room* baze podataka dohvaćaju se računi s kojim korisnik trenutno raspolaže, količina novca s kojom korisnik raspolaže na određenom računu te ukupna količina novca kojom korisnik raspolaže (Slika 3.8.).

```
override fun getAccounts() {
    val accounts = accountRepository.getAllAccounts()
    view.onGetAccounts(accounts)
}

override fun getTotalAccountAmount() {
    val accounts = accountRepository.getAllAccounts()
    var totalAmount = 0F
    accounts.forEach { it: Account
        totalAmount += it.amountOfMoney
    }
    view.onGetTotalAccountAmount(totalAmount)
}
```

Slika 3.8. Funkcije u *AccountsFragmentPresenter* za dohvaćanje računa i ukupne količine novca s kojom korisnik raspolaže

3.5. Pregled – prikaz transakcija na grafu

Transakcije se na grafu prikazuju u fragmentu *OverviewFragment*. Korisnik može odabrati prikaz transakcija primanja i troškova za trenutnu godinu (Slika 3.9.) i trenutni mjesec. Nakon odabira korisnika, transakcije se dohvaćaju iz baze podataka te obrađuju i prikazuju na grafu (Slika 3.10.).

```
private fun getDataForBarChartCurrentYear(typeOfCategory: TypeOfCategory): ArrayList<BarEntry>{
    val categories = categoriesRepository.getAllCategories(typeOfCategory)
    val months = getLabelsForCurYear()
    val yValues = ArrayList<Float>()
    val barEntries = ArrayList<BarEntry>()
    for((br, month) in months.withIndex()) {
        yValues.clear()
        categories.forEach { category :Category ->
            var amount = 0F
            transactionsRepository.getTransactionsByMonthYearCategoryId(category.categoryId!!, monthYearDate: "$month ${getCurrentYear()}")
                .forEach { transaction :Transaction ->
                    amount += transaction.amountOfMoney
                }
            yValues.add(amount)
        }
        barEntries.add(BarEntry(br.toFloat(),yValues.toFloatArray()))
    }
    return barEntries
}
```

Slika 3.9. Funkcija u *OverviewFragmentPresenter* za dohvaćanje podataka za trenutnu godinu

```
override fun getDataForBarChart(barChart: HorizontalBarChart, typeOfCategory: TypeOfCategory, typeData: String) {
    when(typeData){
        TYPE_CURRENT_MONTH-> {
            val entries = getDataForBarChartCurrentMonth(typeOfCategory)
            val stackLabels = getStackedLabelsForHorizontalBarChart(typeOfCategory)
            val colors = getColorsForHorizontalBarChart(barChart, typeOfCategory)
            view.onGetDataForBarChart(entries, stackLabels, colors)
        }
        TYPE_CURRENT_YEAR->{
            val entries = getDataForBarChartCurrentYear(typeOfCategory)
            val stackLabels = getStackedLabelsForHorizontalBarChart(typeOfCategory)
            val colors = getColorsForHorizontalBarChart(barChart, typeOfCategory)
            view.onGetDataForBarChart(entries, stackLabels, colors)
        }
    }
}
```

Slika 3.10. Funckija u *OverviewFragmentPresenter* za dohvaćanje podataka za graf

3.6. Ravnoteža primanja i troškova

Ravnoteža primanja i troškova se nalazi u fragmentu *BalanceFragment* gdje se prikazuje ravnoteža do trenutnog mjeseca u trenutnoj godini. Podatci se dohvaćaju iz baze podataka, obrađuju te prikazuju (Slika 3.11.).

```
override fun getDataForAdapter(){
    val data = ArrayList<Any>()
    val passedMonths = getPassedMonthsForCurrentYear()
    passedMonths.forEach {monthYear : String ->
        val month = getMonthFromMonthYear(monthYear)
        val year = getYearFromMonthYear(monthYear)
        val incomesTotalAmount = getTotalAmountForTypeOfCategory(TypeOfCategory.INCOMES, monthYear)
        val expensesTotalAmount = getTotalAmountForTypeOfCategory(TypeOfCategory.EXPENSES, monthYear)
        data.add(MonthYearBalance(month, year, expensesTotalAmount, incomesTotalAmount))
        data.add(incomesTotalAmount - expensesTotalAmount)
    }
    view.onGetDataForAdapter(data)
}
```

Slika 3.11. Funkcija u *BalanceFragmentPresenter* za dohvaćanje podataka za adapter

4. IZGLED I KORIŠTENJE APLIKACIJE

U ovom djelu će biti prikazano kako izgleda aplikacija bez unesenih podataka i s unesenim podatcima te će biti objašnjeno kako se aplikacija koristi.

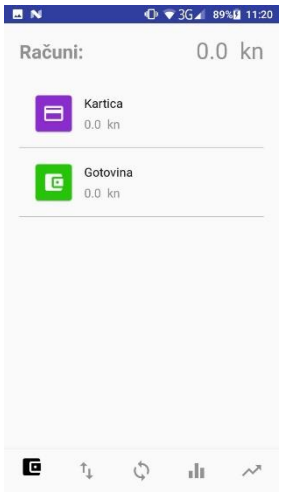
4.1. Bez unesenih podataka

Aplikacija se sastoji od jedne aktivnosti (engl. *Activity*) na kojoj su *BottomNavigation* i *FrameLayout* pomoću kojeg se mijenjaju fragmenti s obzirom na klik na stavku (engl. *item)* na *BottomNavigation*-u. Prilikom ulaska u aplikaciju prikazuju se kategorije troškova i pita (engl. *pie*) graf Sslika 4.1.). Kliznim pomakom prsta po ekranu u lijevu stranu (engl. *swipe to left*) mijenja se fragment i prikazuju kategorije primanja i pita graf (Slika 4.2.). Mijenjanje fragmenata prelaskom prsta po ekranu prema lijevo ili desno je omogućeno pomoću *ViewPagera*. Klikom na sljedeće stavke na *BottomNavigationu* prikazuju se transakcije za odabrano vrijeme (Slika 4.3.), računi (Slika 4.4.), prikaz grafa troškova i primanja (Slika 4.5.) te ravnoteža primanja i troškova za trenutnu godinu do trenutnog mjeseca u godini (Slika 4.6.)

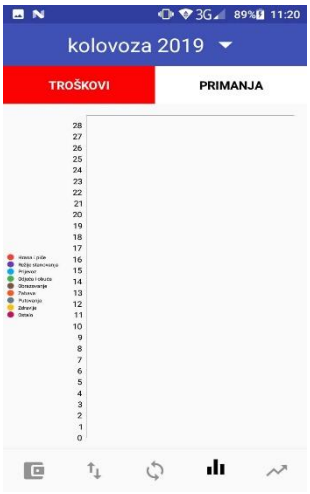
Slika 4.1. Kategorije troškova

Slika 4.2. Kategorije primanja

Slika 4.3. Transakcije



Slika 4.4 Računi



Slika 4.5. Pregled na grafu

Slika 4.6. Ravnoteža

4.2. Unošenje podataka

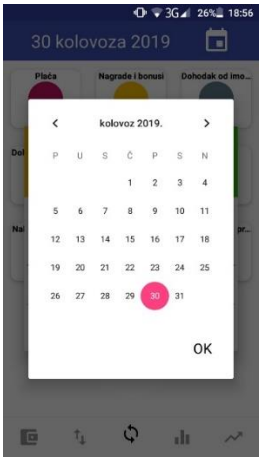
Prilikom pritiska na bilo koju kategoriju od troškova ili primanja otvara se dijalog na kojem se za odabrani dan (Slika 4.9.) može unositi količina novca te bilješka (može se staviti ime potkategorije). Ako se radi o troškovima, prilikom unosa podataka na dijalogu piše „Iz računa“ (koji može biti kartica ili gotovina) i „Za kategoriju“ (koja može biti: hrana i piće, režije stanovanja, prijevoz, odjeća i obuća, obrazovanje, zabava, putovanje, zdravlje i ostalo), (Slika 4.7.). Ako se radi o primanjima onda na dijalogu piše „Iz kategorije“ (koja može biti: plaća, nagrade i bonusi, dohodak od imovine, dohodak od nesamostalnog rada, dohodak od samostalnog rada, mirovine, naknade za nezaposlenost, primljeni pokloni, ostala tekuća primanja) i „Za račun“ (koji može biti kartica ili gotovina), (Slika 4.8.).

Dohodak od nesamostalnog rada uključuje sva primanja iz radnog odnosa iz zemlje i inozemstva, regres za godišnji odmor, naknadu za topli obrok, naknadu za prijevoz, jubilarne i ostale nagrade, posebne primitke u novcu te primitke od rada preko studentskog ili đačkog servisa. Dohodak od samostalne djelatnosti uključuje prihod ostvaren radom u vlastitom obrtu, poduzeću, slobodnom zanimanju, poljoprivrednom gospodarstvu, od autorskih djela, povremenog i privremenog rada, primanja od iznajmljivanja poslovnog prostora, zemlje i pokretne imovine te vrijednost dobara i usluga proizvedenih u vlastitoj proizvodnji i utrošenih ili korištenih u vlastitom kućanstvu. Dohodak od imovine uključuje primanja od dobiti na patente, licence i autorska prava, kamata na štedne uloge, obveznica i drugih vrijednosnih papira, primanja od iznajmljivanja stana, kuće, vikendice, garaže i soba. Naknade vezane za nezaposlenost uključuju naknade za nezaposlenost i prekvalifikaciju te otpremnine i naknade za otkaz. Ostala tekuća primanja

uključuju obiteljske mirovine, primanja vezana za obitelj (dječji doplatak, naknadu za porodni dopust, primanja za opremu novorođenčadi, primanja na ime alimentacije), primanja na teret bolovanja, naknade za tjelesno oštećenje, primanja za zdravstvenu rehabilitaciju, invalidske mirovine, naknadu za troškove stanovanja primljenu od drugih osoba, socijalnu pomoć te stipendije i nagrade za školovanje [13].

Slika 4.7. Dijalog za unos troškova

Slika 4.8. Dijalog za unos primanja



Slika 4.9. Dijalog za odabira datuma

4.3. S unesenim podatcima

U ovom potpoglavlju će biti prikazan izgled aplikacije nakon što je korisnik unio podatke o transakcijama troškova i primanja. Prilikom unosa podataka uneseni su podatci kako bi se vidjela funkcionalnost aplikacije. Ne radi se o stvarnim podatcima tj. o podatcima koji su se unosili kroz određeno vrijeme za stvarno kućanstvo.

4.3.1. Kategorije troškova i primanja

Nakon unosa podataka za troškove ili primanja, ispod svake kategorije se prikazuje ukupna količina novca za odabrani datum. Na pita grafu se prikazuje u obliku postotaka odnos količine novca za sve kategorije gdje su unijeti podatci. Sve to je prikazano na slikama 4.10. i 4.11..

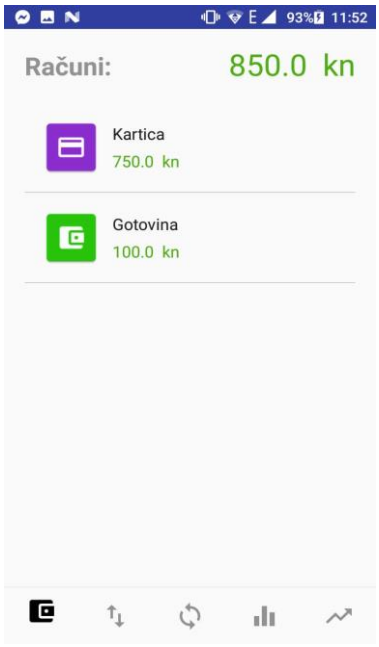
Slika 4.10. Kategorije troškova

Slika 4.11. Kategorije primanja

4.3.2. Računi

Na fragmentu „Računi“ prikazuju se računi korisnika (gotovina i kartica), vrijednost novca kojom trenutno raspolaže pod određenim računom te ukupna vrijednost novca (Slika 4.12.).

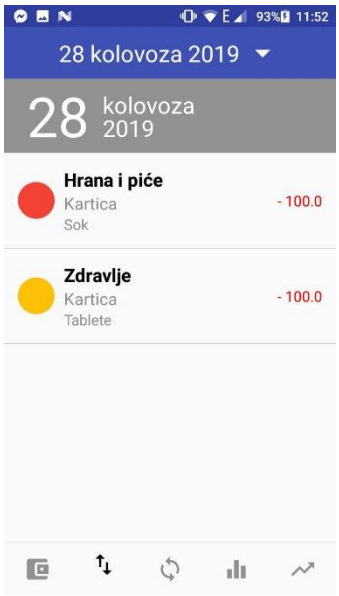
Ako se unese količina novca za troškove, onda se iz ukupne količine novca koju posjeduje odabrani račun oduzima unesena vrijednost (npr. ako je na računu „Kartica“ ukupna količina novca 2000 kn, a potrošilo se 100 kn, onda će ukupna količina novca za račun „Kartica“ biti 1900 kn). Obratno je za primanja, tj. unesena količina novca se zbraja.



Slika 4.12. Računi

4.3.3. Transakcije

Transakcije se mogu vidjeti za trenutni mjesec (Slika 4.15.), trenutni dan (Slika 4.13.), trenutnu godinu (Slika 4.16.) te za cijelo vrijeme (Slika 4.14.). Korisnik sam odabire vrijeme (Slika 4.17.). Nakon odabira vremena od strane korisnika, transakcije se prikazuju poredano s obzirom na datum.

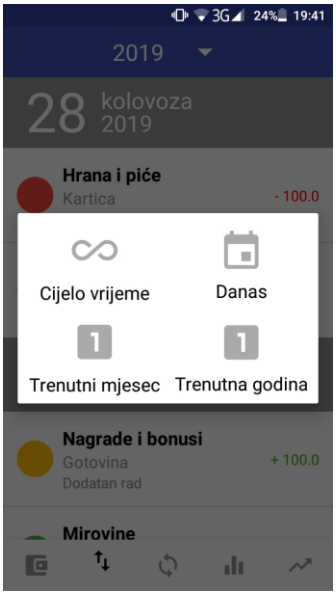


Slika 4.13. Transakcije

Slika 4.14. Transakcije

Slika 4.15. Transakcije

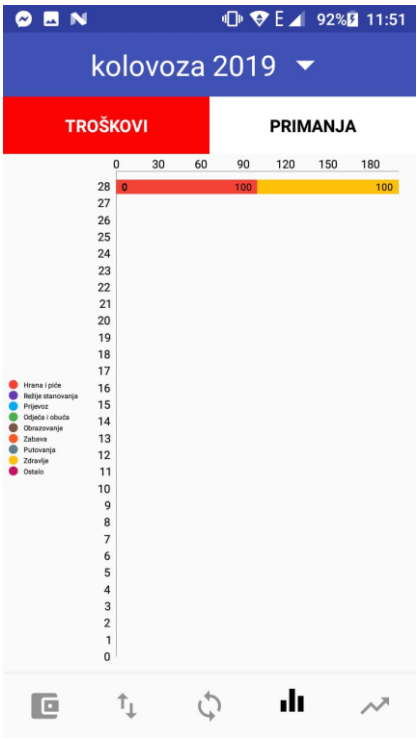
Slika 4.16. Transakcije



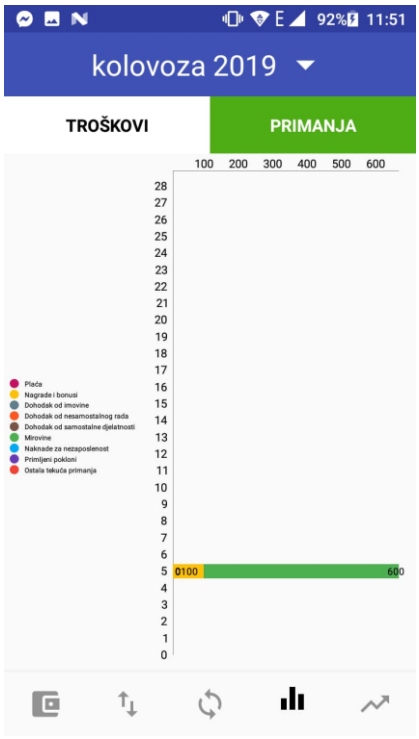
Slika 4.17. Dijalog za odabir vremena

4.3.4. Pregled transakcija na grafu

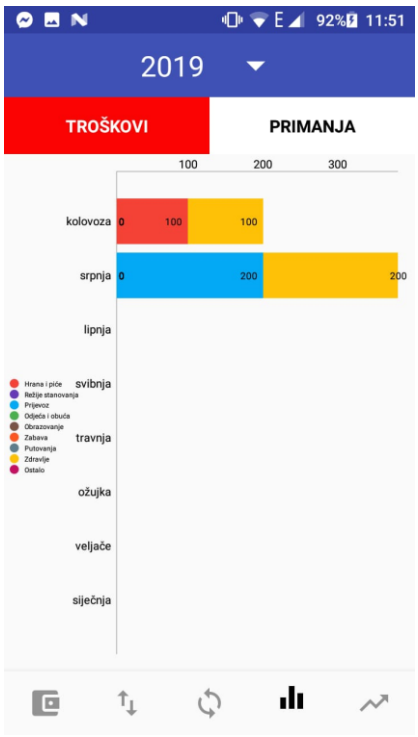
U pregledu se prikazuje graf transakcija za odabrano vrijeme. Korisnik sam odabire vrijeme koje može biti trenutna godina ili trenutni mjesec. Prikazuju se vrijednosti transakcija koje je korisnik unio tijekom odabranog vremena. Svaka kategorija je označena svojom bojom, a pored grafa se mogu vidjeti imena kategorija. Na slici 4.18. se prikazuje graf transakcija troškova za trenutni mjesec. Na slici 4.19. se prikazuje graf transakcija primanja također za trenutni mjesec. Na slici 4.20. se prikazuje graf transakcija troškova za trenutnu godinu. Na slici 4.21. se prikazuje graf transakcija primanja također za trenutnu godinu.



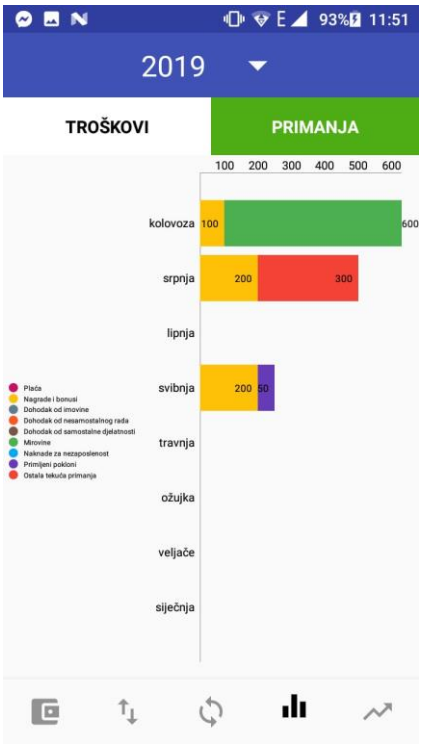
Slika 4.18. Pregled troškova



Slika 4.19. Pregled primanja



Slika 4.20. Prikaz troškova



Slika 4.21. Prikaz primanja

4.3.5. Ravnoteža

Na fragmentu *BalanceFragment* se prikazuje ravnoteža troškova i primanja za trenutnu godinu do trenutnog mjeseca u godini (Slika 4.22.).

Slika 4.22. Ravnoteža

5. ZAKLJUČAK

U radu je opisana Android aplikacija za praćenje troškova kućanstva. Zadatak završnog rada je ispunjen jer aplikacija ispravno radi. Aplikacija nudi razne mogućnosti uvida u troškove i primanja, ali i dalje postoji mogućnost unaprjeđenja aplikacije. Može se dodati mogućnost prikazivanja transakcija, pregleda na grafu i ravnoteže za određeni mjesec određene godine. Mogu se dodati razne druge mogućnosti koje bi aplikaciju dovele na razinu nekih popularnih aplikacija za praćenje troškova.

Prije bilo kakvog pokušaja upravljanja financijama potrebno je napraviti detaljnu analizu trenutne potrošnje, odnosa prihoda i troškova. Ukoliko se ne kontrolira novčani tok unutar kućanstva, teško se može kvalitetno upravljati prihodima i troškovima. Često nismo svjesni gdje trošimo novac, a kako bi se to spriječilo dobro je koristiti aplikacije za praćenje troškova. Ne zahtijevaju puno vremena za unos podataka, a u budućnosti su jako korisne jer imamo uvid u sve transakcije.

Ova aplikacija omogućuje unos prihoda i rashoda za odabrani datum, prikaz rashoda i prihoda za odabrani datum na grafu, podjelu prihoda i rashoda po kategorijama, prikaz transakcija za odabrano vrijeme, prikaz računa kojima korisnik raspolaže, prikaz ukupne svote novca kojom korisnik raspolaže te za svaki račun posebno, prikaz ravnoteže prihoda i rashoda za trenutnu godinu do trenutnog mjeseca. Aplikacija je primjenjiva u stvarnom životu i olakšava praćenje troškova i primanja kućanstva.

LITERATURA

[1] https://developer.android.com/studio/intro, pristupljeno: 27. kolovoza, 2019. godine

[2] https://developer.android.com/guide/topics/manifest/manifest-intro.html, pristupljeno: 27. kolovoza, 2019. godine

[3] https://developer.android.com/guide/topics/resources/providing-resources, pristupljeno: 27. kolovoza, 2019. godine

[4] https://kotlinlang.org/docs/reference/android-overview.html, pristupljeno: 28. kolovoza, 2019. godine

[5] https://developer.android.com/guide/components/activities/activity-lifecycle.html, pristupljeno: 28. kolovoza, 2019. godine

[6] https://developer.android.com/guide/components/fragments.html, pristupljeno: 28. kolovoza, 2019. godine

[7] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView?hl=en, pristupljeno: 29. kolovoza, 2019. godine

[8] https://developer.android.com/reference/android/support/v4/view/ViewPager, pristupljeno: 29. kolovoza, 2019. godine

[9] https://developer.android.com/training/data-storage/room/index.html, pristupljeno: 30. kolovoza, 2019. godine

[10] https://www.tutorialspoint.com/android/android_shared_preferences.htm, pristupljeno: 31. kolovoza, 2019. godine

[11] https://android.jlelse.eu/koin-simple-android-di-a47827a707ce, pristupljeno: 31. kolovoza, 2019. godine

[12] https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/, pristupljeno: 31. kolovoza, 2019. godine

[13] https://www.dzs.hr/hrv/publication/2005/13-2-1_1h2005.htm?fbclid=IwAR1jbVEuMVdFtwyoqa-IshI-JGUe9fskT2ww2VfhuIw0z-KG_wQms0TZkBU, pristupljeno: 2. rujna, 2019. godine

SAŽETAK

U ovom završnom radu obrađena je tema izrade Android aplikacije za praćenje troškova kućanstva. Aplikacija korisniku omogućuje brojne mogućnosti u praćenju troškova i primanja. Prilikom izrade aplikacije primijenjene su razne popularne tehnologije za izradu Android aplikacije. U radu je opisana funkcionalnost aplikacije, dani su primjeri programskog koda. Izrađena aplikacija je testirana na izmišljenim podatcima kako bi se prikazala funkcionalnost aplikacije.

Ključne riječi: kućni budžet, troškovi, primanja, praćenje troškova, praćenje primanja, Android aplikacija

Title of the final paper: Application for tracking household expenses

ABSTRACT

This final article addresses the topic of creating an Android application to track household expenses. The app provides the user with numerous options in tracking expenses and incomes. Various popular technologies were used to create the Android app. The paper describes the functionality of the application, examples of programming code are given. The created application was tested with frame data to show the functionality of the application.

Keywords: home budget, expenses, incomes, cost tracking, incomes tracking, Android app

ŽIVOTOPIS

Igor Kuridža rođen 1. srpnja 1997. godine u Sisku, Hrvatska. Trenutno stanuje na adresi Đure Deželića 53, Selište. 2004. godine započinje osnovnoškolsko obrazovanje u OŠ Mate Lovraka, Kutina. Nakon odličnog uspjeha u osnovnoj školi 2012. godine upisuje Tehničku Školu Kutina, smjer računarstvo. U srednjoj školi sudjeluje na županijskim natjecanjima iz informatike i sportskim natjecanjima. 2016. godine završava srednjoškolsko obrazovanje te upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. 2019. godine uspješno polaže Android akademiju. Trenutno završava treću godinu preddiplomskog studija i planira nastaviti studiranje na nekom od diplomskih studija. Posjeduje vozačku dozvolu B kategorije, diplomu Android akademije, poznavanje programskih jezika C, C++, C# ,Java te Kotlin.

PRILOZI

1. Aplikacija u .docx formatu
2. Aplikacija u .pdf formatu
3. Izvorni kod