Mobilna aplikacija za praćenje osobnih financija

Ključević, Mario

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

MOBILNA APLIKACIJA ZA PRAĆENJE OSOBNIH FINANCIJA

Rijeka, rujan 2022.

Mario Ključević

0069084600

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

MOBILNA APLIKACIJA ZA PRAĆENJE OSOBNIH FINANCIJA

Mentor : doc. dr. sc. Ivan Volarić

Rijeka, rujan 2022.

Mario Ključević

0069084600

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Programiranje**
Grana: **2.03.03 elektronika**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Mario Ključević (0069084600)**
Studij: Preddiplomski sveučilišni studij elektrotehnike

Zadatak: **Mobilna aplikacija za praćenje osobnih financija / Smartphone application for personal financial management**

Opis zadatka:

Korištenjem tehnologije Flutter, potrebno je izraditi višeplatformsku mobilnu aplikaciju za praćenje osobnih financija. Potrebno je omogućiti unos podataka od strane korisnika, gdje za svaku transakciju korisnik unosi iznos, kategoriju, datum i kratki opis transakcije. Također, potrebno je implementirati pregled svih prijašnjih transakcija, te pregled statističkih podataka vezanih za osobne rashode i prihode. Spremanje unesenih podataka potrebno je realizirati pomoću baze podataka Cloud Firestore, pomoću koje će se vršiti autentifikacija korisnika s mogućnošću registracije novih i prijave već postojećih korisnika.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Ivan Volarić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Viktor Sučić

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

IZJAVA

U skladu s člankom 10. Pravilnika o završnom radu i završnom ispitu na preddiplomskim sveučilišnim studijima Tehničkog fakulteta u Rijeci, izjavljujem da sam samostalno izradio ovaj završni rad za mjesec rujan 2022. godine.

Rijeka, rujan 2022.

Mario Ključević

0069084600

ZAHVALA

Htio bih se ovom prilikom zahvaliti za početak svom mentoru doc.dr.sc. Ivanu Volariću na odobravanju ove teme završnog rada koju sam ujedno uzeo i kao hobistički osobni projekt za napredovanje u ovom tehničkom području. Također, zahvaljujem se svojoj obitelji i prijateljima na podršci kroz sve dosadašnje godine studiranja.

SADRŽAJ

1. UVOD

Svjesnost o vlastitim financijama je pojam o kojem bi svi ljudi, a osobito mladi trebali voditi računa. Neodgovornost prema osobnim financijama može dovesti do određenih problema ukoliko se oni ne rješavaju pravovremeno. Ti problemi se vrlo lako mogu izbjeći praćenjem osobnih financija što rezultira time da pojedinac dobiva određenu vrstu kontrole nad svojim troškovima te se dobiva i određena mogućnost uštede. Potaknuto ovom temom, cilj ovog rada je izrada višeplatformske aplikacije za praćenje osobnih financija koja omogućuje pojedincu da u nekoj mjeri, jednostavno i efikasno vodi računa o svom financijskom stanju. Aplikacija je zamišljena na način da korisnik ručno unosi podatke o transakcijama (bile one tipa rashoda ili prihoda) te će sustav pomoću dobivenih informacija o transakcijama prikazivati razne statističke podatke koje će korisniku davati jasan uvid u stanje svojih osobnih financija.

2. MOBILNI OPERACIJSKI SUSTAVI

Operacijski sustav predstavlja softversko sučelje zaduženo za upravljanje hardverskim komponentama te pomaže korisniku u korištenju tih jedinica. Klasični operacijski sustavi korišteni u svakodnevnici mogu se podijeliti na operacijske sustave za osobna računala i mobilne operacijske sustave. Iako većina uređaja koji ne koriste mobilne operacijske sustave imaju dobro napravljene i u praksi testirane alate koje im omogućuju centralno upravljanje, to generalno ne mora vrijediti za mobilne uređaje. Poželjna je mogućnost prilagodbe postavaka standardu koji diktiraju naša osobna pravila te niz drugih funkcija.

Operacijski sustavi za osobna računala, kao što samo ime govori, koriste se u osobnim računalima, bila ona stolna računala ili prijenosna računala. Neki od najpoznatijih su Microsoft Windows, Linux i MacOS. Na drugu ruku, mobilni operacijski sustavi predstavljaju skupinu sustava korištenih u pametnim telefonima, tabletima, pametnim satovima i sličnim drugim elektroničkim uređajima. Ti sustavi rade na nešto drugačijem način u odnosu na one za osobna računala. Oni kombiniraju značajke sustava za osobna računala i drugih korisnih značajki kao što su jednostavnost korištenja na malim ekranima, korištenje SIM kartice za telefoniju i podatkovno povezivanje. Danas postoje mnogi mobilni operacijski sustavi, međutim dva sustava koji dominiraju i najšire prihvaćena su Android (slika 2.1) od tvrtke Google te Apple iOS (slika 2.2). Ova dva sustava funkcioniraju na različite načine.

Slika 2.1 Logo Android OS-a. [6]

Slika 2.2 Logo Apple iOS sustava. [7]

2.1. Android operacijski sustav

Android operacijski sustav je mobilni operacijski sustav razvijen od strane Google-a namijenjen prvenstveno za korištenje na pametnim mobitelima, tabletima te drugim uređajima s dodirnim zaslonom. Njegov dizajn omogućuje korisnicima da intuitivno manipuliraju mobilnim uređajima pokretima prstiju koji odražavaju uobičajene pokrete kao što su tipkanje, povlačenje i druge slične radnje. Google pristupa razvoju Androida na bazi otvorenog koda (eng. *Open source*) što znači da je kod sustava javno dostupan te svaki pojedini proizvođač uređaja može prilagoditi sustav svojem uređaju po vlastitoj želji. Od svog lansiranja 2007., Android je postao najzastupljeniji sustav za pametne telefone. Jezgra Androida bazirana je na Linuxu te na programskog jeziku Java.

Dostupnost preko tri milijuna različitih Android aplikacija na službenom repozitoriju *Google Play Store* i ne previše strogo kontrolirano okruženje za razvojne programere aplikacija, doprinijeli su popularnosti Androida i porastu broja Android uređaja. To je ujedno i najveća prednost Androida nad ostalim sustavima. Međutim, upravo ta otvorenost sustava pružila je priliku programerima zlonamjernog softvera da upadnu u Android uređaje putem zlonamjernih aplikacija koje se mogu naći u raznim trgovinama aplikacija, između ostalog i same službene *Google Play Store* trgovine. Takve aplikacije mogu omogućiti pristup trećim osobama različitim vrstama privatnih i osjetljivih podataka kao što su SMS poruke, pozivi, elektronička pošta, slike, kontakti, lokacije, lozinke itd. Zlonamjerno korištenje takvih podataka može dovesti do prijevare, prijetnji te samog financijskog gubitka.

2.2. iOS operacijski sustav

Nakon Androida, kao drugi najzastupljeniji mobilni operacijski sustav je i iOS. Službeno je predstavljen 2007. na prvoj generaciji Apple iPhone uređaja. Razvijen je od strane Applea, baziran na MacOS-u te je kao sustav podržan jedino na samim Apple uređajima. Time je Apple postigao mogućnost regulacije svih aplikacija i servisa koji se mogu pokretati na iOS uređajima. Razvoj aplikacija za iOS moguće je ostvariti putem Appleovog alata Xcode i programskog jezika Swift (u prošlosti Objective-C), međutim, dalje u radu bit će objašnjeni i drugi načini razvoja mobilnih aplikacija. iOS je u odnosu na Android potpuno suprotan u smislu prava na korištenje. Kao što je već navedeno, iOS radi samo na Apple mobilnim uređajima što već u startu kontradiktira Androidu koji je tipa otvorenog koda. Ta značajka može otežavati posao razvojnim programerima aplikacija za iOS iz razloga što Apple nameće svoja stroga pravila koje aplikacije moraju zadovoljavati ukoliko žele biti dostupne na službenom repozitoriju aplikacija App Store. Takva

vrsta „zatvorenosti“ i „strogosti“ sustava od strane Apple-a ne mora se nužno predstavljati samo kao loša stvar. Upravo suprotno, takva politika može dovesti do toga da su uređaji, odnosno njihov hardver, koji pokreću takav sustav izvanredno optimizirani za isti. Fizikalni razlog tome je taj da je iOS namijenjen izričito radu na malom broju različitih uređaja, odnosno samo na Apple pametnim telefonima, što čini optimizaciju operacijskog sustava jednostavnijom. Dok je na drugu stranu, Android, koji radi na ogromnom broju pametnih telefona, gotovo nemoguće optimizirati da jednako dobro radi na svakom hardveru u svakom uređaju koje proizvode različite tvrtke.

Osim iOS-a, Apple je na temelju njega izradio i druge mobilne sustave, namijenjene sličnim pametnim uređajima isto tako striktno vezanih uz Apple. Neki od njih su iPadOS za tablet uređaje i watchOS za pametne satove.

3. NATIVNI I VIŠEPLATFORMSKI RAZVOJ APLIKACIJA

Danas postoji više načina izrade mobilnih aplikacija. Jedan od načina je tzv. nativni ili izvorni razvoj. Nativni način podrazumijeva izradu aplikacija za strogo određene mobilne operativne sustave kojima korisnici pristupaju iz odgovarajućeg repozitorija aplikacije (npr. *Google Play Store* na Androidu i *App Store* na iOS-u). Ukoliko je namjera izrada aplikacije za Android, razvojni programeri koristit će programske jezike Java ili Kotlin (noviji programski jezik, usko vezan uz Javu). Na drugu ruku, razvoj za iOS uređaje vrše se putem programskih jezika Objective-C ili Swift (Appleov vlastiti jezik). I Google i Apple razvojnim programerima aplikacija nude vlastite razvojne alate, elemente sučelja i SDK[1].

Na drugu stranu, osim nativnog razvoja postoji i višeplatformski razvoj (eng. *Cross-Platform Development*). Takva vrsta razvoja prakticira razvijanje aplikacija ili servisa za više platforma ili okruženja odjednom što u prijevodu znači da jedan programski kod pokriva dvije ili više ciljane platforme. Već tu je vidljiva najveća pogodnost višeplatformskog razvoja u smislu smanjenja potrebe za razvojem više zasebnih baza koda. Upravo ovaj način razvijanja mobilnih aplikacija preko određenih tehnologija će biti primijenjen u sklopu ovog rada.

3.1. Generalne razlike između nativnih i višeplatformskih aplikacija

I nativne i višeplatformske razvojne tehnologije u stalnom su stanju evolucije. Ovakva promjenjiva priroda tehnologija signalizira da se ove teme s vremena na vrijeme treba ponovno proučiti kako bi se razumjelo koja od ovih opcija trenutno vodi u svijetu razvoja softvera. Razvoj nativnih aplikacija izbjegava složnost stvaranja održivog proizvoda koji obuhvaća razvoj aplikacija na više platformi i usredotočuje se na generiranje kompetentnog dizajna koji ostaje blizu ciljne platforme . Višeplatformske tehnologije nastoje generirati aplikaciju koja dopire do što većeg broja korisnika pokrivajući široki broj krajnjih uređaja tijekom procesa dizajna i implementacije. U tablici 3.1 prikazane su neke od glavnih razlika između ove dvije razvojne tehnologije. Dvije glavne značajke koji bi se mogle izdvojiti su cijena razvoja i upotrebljivost koda koje su ujedno i međusobno uzročno-posljedične. Kako bi se željeni projekt realizirao u nativnom načinu za dvije platforme, posljedično tome potrebno je pisati dva razvojna koda što uvelike povećava troškove realizacije projekta dok suprotno tome, višeplatformski način cijenu praktički

[1] SDK (eng. *Software Development Kit*) – Skup alata za razvojne programere za korištenje u proizvodnji aplikacija koje koriste određenu platformu

smanjuje skoro upola. Samim time, jednu bazu koda jednostavnije je održavati te je lakše i brže izbacivati ažuriranja softvera što je velika prednost.

Međutim, višeplatformski razvoj aplikacija ima i svoje nedostatke što je i logično jer bi u protivnom u potpunosti preuzeo razvojno tržište. Ako traženi projekt zahtjeva izrazitu robusnost i veliku potrebu za korištenjem određenih hardverskih dijelova uređaja kojem je program namijenjen (velika točnost GPS-a, upotreba kamere, senzora i sličnih drugih hardverskih dijelova), nativna aplikacija je puno pouzdaniji i bolji izbor. Kod analize performansa, u nekim slučajevima (kod kompleksnijih aplikacija) dolazi do malog pada performansi tako razvijene aplikacije.

Tablica 3.1 Razlike nativnih i višeplatformskih aplikacija.

Parametar	Nativna aplikacija	Višeplatformska aplikacija
Cijena	Visoka cijena razvoja	Relativno niska cijena razvoja
Upotrebljivost koda	Radi samo na jednoj platformi	Jedna baza koda za više platformi
Pristup uređaju	Platformski SDK osigurava pristup API[2]-ju uređaja bez ikakvih prepreka	Nema zajamčenog pristupa svim API značajkama uređaja
Konzistencija korisničkog sučelja	Konzistentnost sa elementima korisničkog sučelja uređaja	Ograničena konzistencija sa elementima korisničkog sučelja
Performanse	Odlične performanse, s obzirom da je aplikacija razvijena isključivo za taj određeni sustav	Visoke performanse, međutim mala štekanja i problemi sa kompatibilnošću hardvera nisu rijetka.

[2] API (eng. Application Programming Interface) – Softverski posrednik koji omogućava dvjema aplikacijama međusobnu komunikaciju

3.2. Izazovi i pogodnosti višeplatformskog mobilnog programiranja

Prije nekoliko godina razvoj aplikacija na više platformi bio je ograničen na izradu jednostavnih mobilnih aplikacija i igara. S vremenom su nove tehnologije učinile razvoj na više platformi prilagodljivim, snažnijim i fleksibilnijim nego prije. Unatoč tome, višeplatformski razvoj se i dalje suočava s izazovima kao što su:

- Poteškoće u izvedbi zbog nedosljedne komunikacije između izvornih komponenti uređaja.
- Poteškoće s održavanjem unakrsne usklađenosti aplikacija na različitim uređajima i operacijskim sustavima.
- U nekim slučajevima greške povezane s performansama mogu dovesti do lošeg korisničkog iskustva.
- Ako poslovna aplikacija upravlja dijelom tvrtke i pohranjuje korisničke podatke, višeplatformski razvoj nije uvijek dobra ideja zbog sigurnosnih razloga.

Međutim, unatoč navedenim manama postoji i veliki broj prednosti višeplatformskog razvoja kao što su:

- Maksimalna izloženost ciljanoj publici
- Smanjeni troškovi proizvodnje
- Lakši razvoj i održavanje
- Brži razvojni proces

3.3. Flutter razvojni okvir

Kroz ovaj rad, pozornost je posvećena višeplatformskom razvoju te je tako i sama aplikacija za praćenje osobnih financija izvedena u tom načinu. Razvoj se vršio putem tehnologije Flutter, odnosno točnije rečeno, Flutter razvojnog okvira (eng. *framework*). Razvojni okvir u programiranju je vrsta apstrakcije u kojoj se softver, koji pruža generičku funkcionalnost, može selektivno mijenjati dodatnim korisničkim kodom, čime se osigurava softver specifičan za određenu aplikaciju. Drugim riječima, on pruža standardni način za izgradnju i implementaciju aplikacija i univerzalno je softversko okruženje koje pruža određenu funkcionalnost kao dio veće softverske platforme za olakšavanje razvoja softverskih aplikacija, proizvoda i sličnih rješenja. Razvojni okviri mogu uključivati kompajlere, biblioteke kodova, programe podrške, skupove alata i sučelja za programiranje aplikacija (API) koji spajaju sve različite komponente kako bi omogućili

razvoj projekta ili sustava. Oni u suštini pružaju gotove komponente ili rješenja koja su već prilagođena kako bi se sam razvoj i implementacija ubrzala.

Flutter (slika 3.2) je Google-ov besplatni okvir korisničkog sučelja otvorenog koda za izradu izvornih mobilnih aplikacija, objavljen 2017. godine. On omogućuje programerima mobilnih aplikacija da s jednom bazom koda i jednim programskim jezikom izrađuju višeplatformske aplikacije te time uvelike olakšava i ubrzava razvoj softvera za Android i iOS. Flutter je kompletno baziran na principu korištenja *widgeta* u razvoju. *Widget (slika 3.1)* predstavlja određeni element korisničkog sučelja koji se pojavljuje na ekranu te se uvelike oslanja na principe objektno-orijentiranog programiranja.

```
child: Column(
  children: [
    Row(
      children: [
        const CircleAvatar(
          backgroundColor: Colors.black,
          child: Icon(
            Icons.person,
            color: Colors.white,
          ), // Icon
        ), // CircleAvatar
        const SizedBox(width: 10),
        Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(user!.name,
                style: Theme.of(context).textTheme.titleLarge), // Text
            Text(user.email,
                style: Theme.of(context).textTheme.labelSmall), // Text
          ],
        ), // Column
      ],
```

Slika 3.1 Primjer korištenja widgeta u razvoju.

Kao programski jezik koristi se Dart, objektno orijentiran, usredotočen na *front-end*[3] razvoj sa svojom vrlo jednostavnom i intuitivnom sintaksom, sličnom onoj kod JavaScript programskog jezika. Flutter je još relativno novi višeplatformski *framework*, međutim njegove pogodnosti sve više privlače razvojne programere naspram sličnih višeplatformskih opcija kao što su React Native, Cordova, Xamarin i dr. Prije svega, Flutter je jednostavan za naučiti zato što nije potrebno pisanje velikog koda te sadrži već gotove standardne *widget*e za ciljanje platforme što sam proces razvoja čini puno jednostavnijim i bržim. Osim ovih razloga, jedan od najvećih prednosti Flutter-a je izvanredna dokumentacija i tzv. *community*, odnosno zajednica ljudi, programera koji aktivno koriste određeni programski jezik ili tehnologiju te time pridonose popularnosti, poboljšanju i

[3] Eng. *Front-end* razvoj - Pojam u programiranju koji se odnosi na razvoj softvera vezanog isključivo za korisničko sučelje, odnosno svega onog „okom vidljivog“ na aplikaciji, web stranici i sl.

općom dostupnosti informacija. Flutter posjeduje veliku online bazu resursa koja daje odgovore na sva razna pitanja koja bi si programer mogao postaviti tijekom razvoja aplikacija u Flutteru. To je upravo tako zahvaljujući sjajnoj dokumentaciji koju je Google-ov tim složio i objavio na službenim stranicama s jednostavnim sučeljem. A ako i sva već postojeća dokumentacija ne pomogne, programeri uvijek mogu pronaći rješenja i razmjenjivati ideje na mjestima kao što su Flutter Community i Flutter Awesome.

Zaključno rečeno, Flutter razvojni okvir tvrtkama i njihovim razvojnim programerima predstavlja intuitivna i jednostavna rješenja za izradu višeplatformskih aplikacija s jednom bazom koda na sučelju vrlo jednostavnom za korištenje te čini razvoj mobilnih aplikacija brzim i isplativim.

Slika 3.2 Logo Flutter razvojnog okvira. [8]

4. RAZVOJ APLIKACIJE ZA PRAĆENJE OSOBNIH FINANCIJA

Kako je već u uvodu rečeno, cilj ovog rada je izrada višeplatformske aplikacije za praćenje osobnih financija te predstavljanje samog toka razvoja. Aplikacija je zamišljena kao rješenje za pojedinca koji želi steći uvid u bilancu svojih prihoda i rashoda. Za sam početak, potrebno je realizirati mogućnost registracije i naknadne prijave korisnika u aplikaciju putem adrese elektroničke pošte i lozinke. To se vrši putem zasebnog ekrana za registraciju i prijavu korisnika. Nakon uspješne registracije ili prijave, aplikacija učitava glavni te ujedno i početni zaslon koji okvirno prikazuje neke od pojedinačnih aspekata statističkog praćenja u okviru kartica. Na tom zaslonu korisnik dobiva opći sažeti uvid u neke od raznih statističkih podataka. Osim početnog zaslona, kao jedan od bitnijih zaslona je i onaj za pregled svih, već obavljenih transakcija. U njemu se, u obliku liste prikazuju sve već ostvarene transakcije zajedno sa njihovim obilježjima kao što su ime kategorije, iznos transakcije, vrsta transakcije (prihod ili rashod), datum te kratki opis. Ujedno, postoji i mogućnost filtriranja pregleda transakcija po vremenskom kriteriju, kategoriji transakcije ili pretrazi ključnih riječi. Isto tako, važno je naglasiti kako je kroz cijelu aplikaciju moguća, putem glavne donje navigacijske trake, navigacija kroz glavne zaslone aplikacije. U navigacijskoj traci, nalazi se i centralna akcijska tipka koji pritiskom vodi korisnika na zaslon za unošenje nove transakcije. Zaslon za dodavanje novih transakcija se sastoji od elemenata za unos i odabir podataka te glavne akcijske tipke za pohranu same transakcije. Osim navedenih zaslona, u donjoj navigacijskoj traci nalazi se i opcija za prelazak na zaslon vezan za cjelokupnu statistiku kroz koji je moguće dobiti uvid u detaljniju statistiku ostvarenog financijskog stanja. Te za kraj, prisutan je i zaslon postavki. Svi ostali detalji i zasloni opisani su u nadolazećim poglavljima.

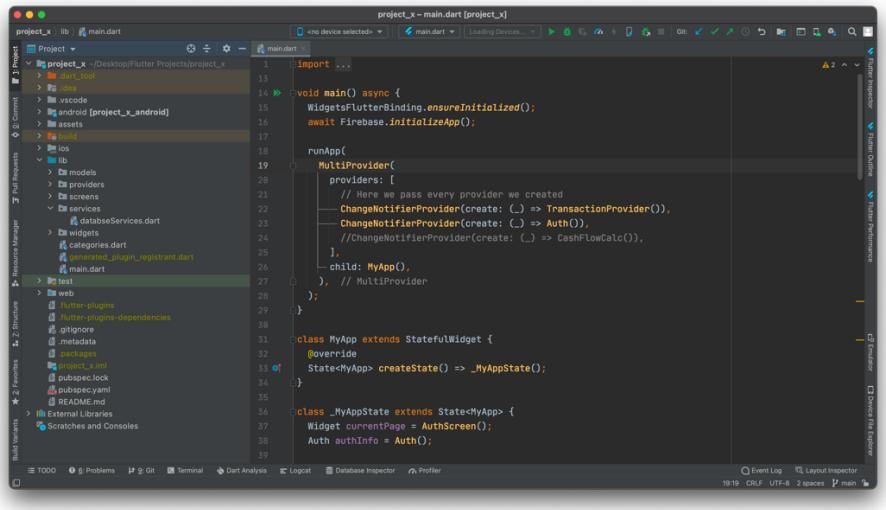
4.1. Razvojno okruženje i priprema alata

Prije pristupa samom programiranju aplikacije i implementiranju rješenja, potrebno je pripremiti okruženje za rad i podesiti alate potrebne za realizaciju projekta.

4.1.1. Integrirano razvojno okruženje

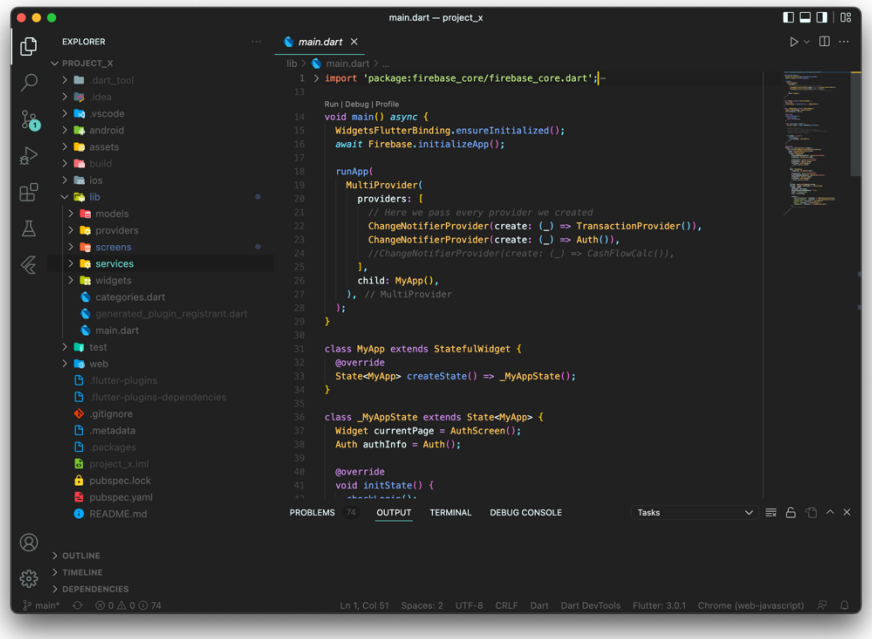
Za potrebe razvoja aplikacija u Flutter razvojnom okviru, neophodno je imati program koji će sadržavati alate potrebne za pisanje samog koda, testiranja, pregled i sl. Drugim riječima, potrebna je neka vrsta integriranog razvojnog okruženja. Integrirano razvojno okruženje predstavlja paket koji objedinjuje osnovne alate potrebne za pisanje i testiranje softvera. U osnovi, ono objedinjuje različite aspekte računalnog programa u jedno grafičko korisničko sučelje. Tijekom razvoja softvera, razvojni programeri koriste nekoliko različitih alata, uključujući uređivač teksta, kompajlere, sustave za otklanjanje grešaka i sl. Bez razvojnog okruženja, ovim alatima se mora upravljati zasebno.

Kao jedan od najpoznatijih integriranih razvojnih sučelja za mobilno programiranje je Android Studio (slika 4.1). Sadržava sve alate potrebne za razvoj mobilnih aplikacija te je generalno najpopularnija opcija za razvojne programere koji žele pristup svim potrebnim pa čak i onim naprednijim alatima za razvoj mobilnih aplikacija. Međutim, Android Studio dolazi i s par mana koje će možda određene programere prisilite na korištenje nekih drugim razvojnih sučelja. Jedna od mana Android Studia je zahtjev samog programa za snažnijim računalnim hardverom i većom količinom resursa prilikom rada programa. Android Studio je veliki program koji dolazi s brojnim alatima koji od računala na kojem programer radi traži mnogo. Stoga, ako računalo ne zadovoljava preporučene sistemske zahtjeve za rad, sam tok razvoja bi mogao biti usporen i pomalo nezgrapan zbog nedostatke resursa koje računalo može dati programu. Drugi razlog zbog kojeg se razvojni programeri često okreću alternativnim razvojnim okruženjima je kompleksnost paketa Android Studio. Paket, kao što je već rečeno, dolazi s jako puno različitih alata koji razvojni programeri mogu koristiti prilikom realizacije željenog projekta. Međutim, svima ne treba uvijek sve. Iz tog razloga, tijekom realizacije nekih projekata, nema potrebe za korištenjem svih mogućih alata koje Android Studio nudi te oni samo usporavaju tok rad i troše dragocjene resurse računala.



Slika 4.1 Sučelje Android Studio paketa prilikom razvoja aplikacije.

Osim Android Studio razvojnog sučelja, tu je još par drugih programa koji nude slične alate. Međutim, program koji je odabran za realizaciju ovog zadatka je Visual Studio Code ili kraće VS Code (slika 4.2). Za razliku od Android Studia, koji je puno integrirano razvojno kruženje, VS Code je uređivač koda (eng. *Source code editor)*. To znači da ne sadrži mnoge razne alate nego se fokusira na jednostavnost i performanse. Još jedna velika razlika je i u mogućnosti prilagođavanja u vidu mijenjanja tema boja uređivača koda te raznoraznih ekstenzija koje se mogu dodatno instalirati kako bi iskustvo korištenja i efikasnost bila bolja.



Slika 4.2 Sučelje VS Code uređivača koda prilikom razvoja aplikacije.

4.1.2. Uređaji za testiranje aplikacije

Cijeli smisao izrade mobilne aplikacije ne bi imao smisla ako ne postoji način za vizualizaciju onoga što se radi. Drugim riječima, potrebno je imati uređaj na kojem će se tijekom cijelog razvoja pratiti i testirati promjene koje su odraz promjene samog koda u razvojnom okruženju. U tu svrhu može se koristiti pravi fizički mobilni uređaj, koji je u tom slučaju potrebno kablom spojiti na računalo ili virtualni mobilni uređaj (slika 4.3) (eng. *Emulator*) koji se pokreće preko računala. Oba načina imaju svoje prednosti i mane, ali za optimalan razvoj potrebno je koristiti i fizičke i virtualne uređaje. Tijekom samog razvoja, jednostavnije je koristiti virtualni uređaj iz razloga što eliminira potrebu za pravim fizičkim uređajem pri ruci. Na drugu ruku, fizički uređaji igraju najveću ulogu kada je potrebno testiranje aplikacije na stvarnom uređaju. Jedino preko fizičkih uređaja je moguće testirati stvarne performanse aplikacija. Isto tako, ako aplikacija koristi primjerice značajke kamere ili lokaciju, na virtualnom uređaju je to teško pa čak i nemoguće simulirati. Kroz projekt korištena su dva tipa emulatora: Google Pixel 6 (Android 12.0) i iPhone 12 (iOS 15.5).

Slika 4.3 Emulator iPhone 12 uređaja.

4.1.3. Sustavi kontrole verzija

Tijekom razvoja softvera, jako je poželjno imati određenu mogućnost praćenja promjena u kodu, mogućnosti suradnje s drugim razvojnim programerima, pregleda prijašnjih verzija koda i sl. U tu svrhu, koriste se sustavi kontrole verzija. Kontrola verzija isto poznata i kao kontrola izvora je praksa praćenja i upravljanja promjenama softverskog koda. Sustavi kontrole verzija softverski su alati koji pomažu timovima razvojnih programera da upravljaju promjenama izvornog koda tijekom vremena te tako omogućuju brži i pametniji rad na projektima. Softver za kontrolu verzija prati svaku promjenu koda u posebnoj vrsti baze podataka. Ako se napravi pogreška, razvojni programeri mogu vratiti vrijeme i usporediti ranije verzije koda kako bi ispravili pogrešku, a minimizirali ometanje svih ostalih članova tima.

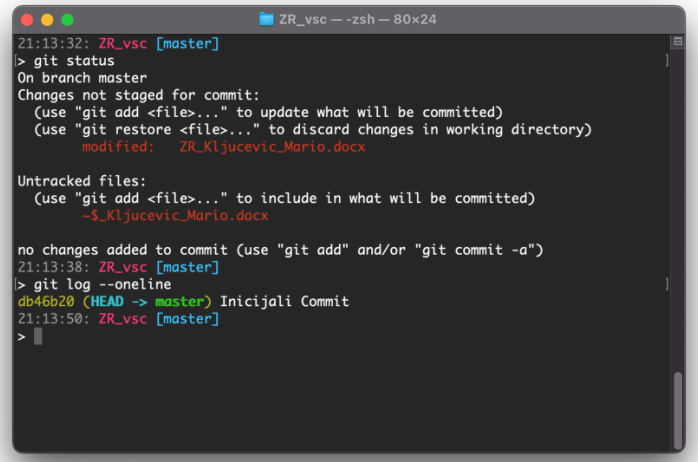
Neke od primarnih prednosti koje se mogu očekivati od kontrole verzija su sljedeće:

1. Potpuna dugoročna pohrana svih promjena svake datoteke. Promjene uključuju stvaranje i brisanje datoteka kao i uređivanje njihovog sadržaja. Ova povijest također uključuje autora, datum i pisane bilješke o svrsi svake promjene. Posjedovanje ove potpune povijesti omogućuje povratak na prethodne verzije u različite svrhe, npr. analizi uzroka greške.

2. Grananje i spajanje. Grananje omogućuje različitim članovima tima da rade istodobno na različitim aspektima softvera. Stvaranje tzv. grana u alatima kontrole verzija održava višestruke tokove rada neovisnim jedna o drugome, a istovremeno pruža mogućnost ponovnog spajanja tog rada, omogućavajući programerima da provjeru kako međusobne grane ne bi bile u sukobu. Generalno je usvojena praksa grananja za svaku odvojenu značajku softvera, novo izdanje ili oboje istovremeno.

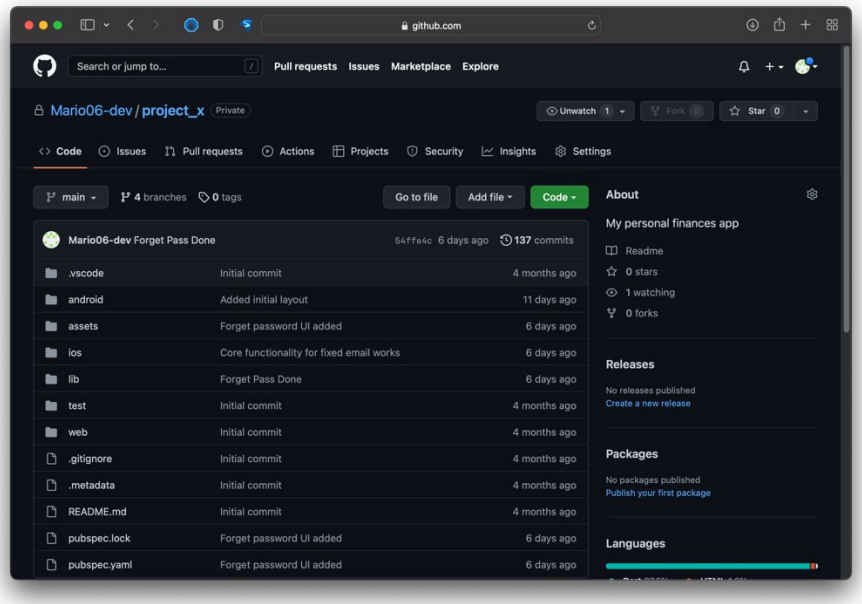
3. Praćenje. Mogućnost praćenja svake promjene napravljene u softveru te mogućnost označavanja svake promjene porukom koja opisuje svrhu i namjenu same te promjene uvelike pomaže u analizi izvora pogreške.

Sustav za kontrolu verzija korišten u ovom radu, a ujedno i trenutno najpopularniji sustav kontrole verzija u svijetu je Git (slika 4.4). Originalno izrađen od strane tvorca Linux operativnog sustava, Linus Torvalda 2005. godine. Git je primarno namijenjen radu u sučelju naredbenog retka (eng. *CLI - Command Line Interface*) zbog efikasnosti korištenja, ali postoji i inačica s grafičkim korisničkim sučeljem za lakšu vizualizaciju raznih aspekata. Git sam za sebe izvršava se lokalno na instaliranom računalu te nije potrebno imati internetsku vezu. Kao baza Git-a, tu je git repozitorij. On prati i sprema povijest svih promjena napravljenih u datotekama u Git projektu. Te podatke sprema u direktorij pod nazivom *.git*, također poznatoj kao repozitorij mapa. Na samom početku projekta, inicijalizira se prazni git repozitorij naredbom *git init.* Nakon toga, svaka datoteka koja se pronađe unutar te mape bit će podložna praćenju promjena unutar nje. Nadalje, korištenjem brojnih drugih naredba, moguće je manipulirati datotekama te imati uvid u prijašnje promjene.



Slika 4.4 Sučelje Git sustava za kontrolu verzija u terminalu.

Osim lokalnog aspekta sustava za kontrolu verzija, gdje se sve promjene, radnje i pregledi vrše lokalno na računalu pojedinca, postoji i mogućnost objavljivanja lokalnog git repozitorija na mrežu. Time se postiže jedan novi nivo međusobne suradnje više ljudi na istom projektu te se ujedno dobiva i mogućnost sigurnosne kopije kao dodatne razine sigurnosti. Ovaj mrežni pristup omogućen je putem udaljenih repozitorija. Oni predstavljaju repozitorije, iste kao i one lokalne, samo objavljene na internetu na određenim stranicama namijenjenima za rad s udaljenim repozitorijima. Kao najpopularnija takva stranica je Github. Ovaj se pojam, kod početnika koji tek ulaze u svijet kontrole verzija, često miješa s pojmom Git, međutim oni predstavljaju dvije različite stvari. Git, kao što je već opisano, predstavlja lokalni sustav, odnosno naredbeni jezik za upravljanje i praćenje promjena u lokalnom git repozitoriju dok je Github internetska stranica za spremanje i pregled udaljenih repozitorija. Github (slika 4.5) omogućuje prenošenje i sinkronizaciju lokalnih git repozitorija u udaljene repozitorije, čime se otvaraju nove mogućnosti kolaboracije više ljudi na jednom projektu. Korisnik može, ukoliko je udaljeni repozitorij otvoren za javnost, jednostavno kopirati već postojeći repozitorij objavljen na Github-u te ga prenijeti lokalno na svoje računalo na kojem može dalje sudjelovati u radu na određenom projektu te naknadno, uz dozvolu administratora, ponovno sinkronizirati na udaljeni repozitorij.



Slika 4.5 Sučelje Github stranice tijekom aktivnog razvoja aplikacije.

4.1.4. Firebase web servis

Kako bi aplikacija imala određene mogućnosti kao što je primjerice mogućnost sinkronizacije na više uređaja, potrebno je implementirati neku vrstu internetske baze podataka i registraciju ili prijavu korisnika u aplikaciju. Da bi se pojednostavio taj cjelokupan proces, mogu se koristiti određeni servisi koji već u sebi sadrže ukomponirane razne alate. Jedan od tih servisa je Google-ov Firebase. Firebase sadrži skup alata koji pokrivaju veliki dio usluga koje bi razvojni programeri inače morali sami izraditi kako bi ih mogli koristiti. To uključuje stvari kao što su baza podataka, analitika, pohrana datoteka, autentifikacija i itd. Dva najkorisnija alata korištena u ovom radu su *Cloud Firestore* baza podataka i *Firebase Authentication.* Za pružanje usluga internetske baze podataka zadužen je *Cloud Firestore.* On uvelike olakšava pristup podacima, uklanjajući potrebu za posebnim posrednikom između aplikacije i baze podataka, kao što je stvar kod standardnih baza podataka. Važno je naglasiti kako je ovo ne-relacijska baza podataka. Osim ne-relacijski postoje i relacijske baze podataka. Isto tako, ove baze podataka su često poznate i kao SQL (relacijske) i NoSQL (ne-relacijske) baze. Ime su dobile ovisno o tome koristi li se u njima SQL[4] jezik ili ne.

[4] SQL (eng. Structured Query Language) - Strukturni jezik za pristupanje i manipuliranje bazama podataka

Neke od glavnih razlika između ove dvije vrste baza podataka prikazane su u tablici 4.1:

Tablica 4.1 Razlike relacijskih i ne-relacijskih baza podataka.

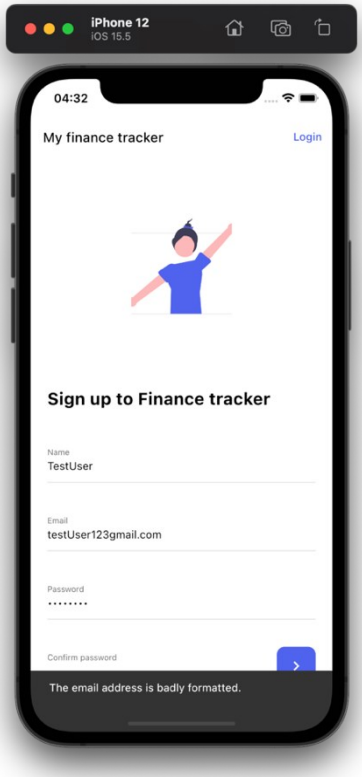
Relacijske baze	**Ne-relacijske baze**
Unaprijed definirana shema	Dinamička shema za nestrukturirane podatke
Vertikalno skalabilne	Horizontalno skalabilne
Bazirane na tablicama	Bazirane na principu „ključ-vrijednost“
Moguće definirati odnose među elementima baze	Nije moguće definirati međusobne odnose među elementima baze

Kao što je prije rečeno, u ovom radu je također korišten *Firebase Authentication* pomoću kojeg se vrši prijava i identifikacija korisnika aplikacije. Autentifikacija je potrebna kako bi se ograničio pristup podacima korisnicima za koje oni nisu prvobitno namijenjeni. Ono što *Firebase Authentication* uvelike posebno olakšava je izvođenje sigurnih prijava, što je izrazito teško samostalno ispravno implementirati. Moguća je implementacija prijave korisnika putem raznih načina uključujući klasičnu prijavu putem elektroničke pošte i lozinke (metoda korištena u radu), prijava putem Google računa, Facebook računa i sl.

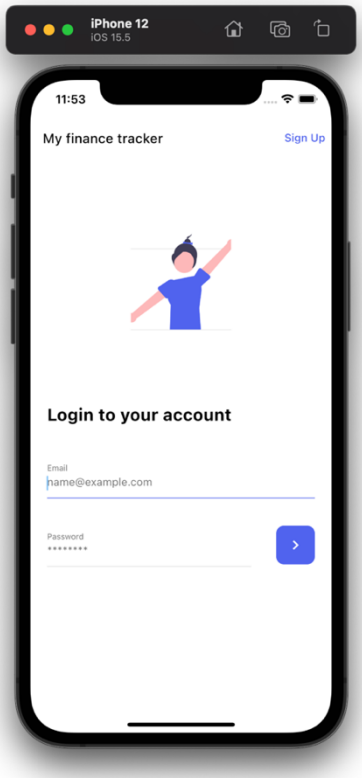
4.2. Implementacija registracije i prijave korisnika

Nakon uspješnog postavljanja svi alata potrebnih za rad na izradi aplikacije, može se krenuti u aktivni razvoj zadatka ovog rada, odnosno mobilne aplikacije za praćenje osobnih financija. Aplikacija je zamišljena tako da sadrži mogućnost registracije novih korisnika te prijave već postojećih putem elektroničke adrese i lozinke. Time je kao logičan početak u razvoju ovakvog tipa aplikacije upravo implementacija autentifikacije korisnika. Sustav autentifikacija u ovom radu, kao što je već opisano u prethodnom dijelu, vrši se putem usluga Firebase web servisa.

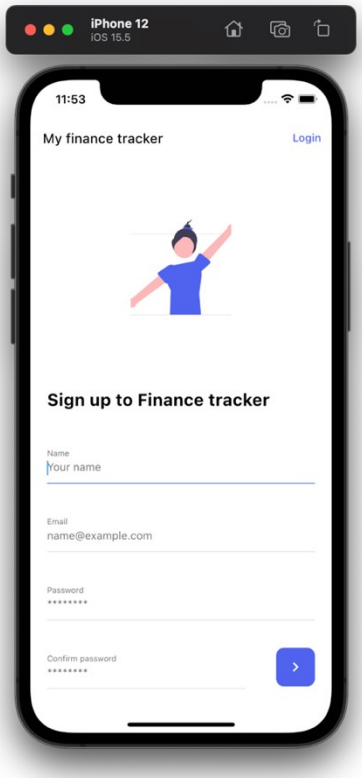
Kao što je prikazano na slikama 4.6 i 4.7. prijava i registracija korisnika vrši se putem dva zasebna zaslona. Zaslon za prijavu korisnika u aplikaciju, ujedno i prvi zaslon s kojim se novi korisnik aplikacije susreće, funkcionalno sadrži dva tekstualna polja za unos podataka, tipka za prijavu, tzv. akcijska tipka, te posebna tekstualna tipka u zaglavlju zaslona kojom korisnik navigira na zaslon za registraciju. Dodatno tome, u zaslonu za registraciju, nalaze se dva dodatna tekstualna polja, jedno za unos imena korisnika, drugo za potvrdu lozinke koja se postavlja za račun. Polja za unos lozinke, tijekom unosa teksta sakrivaju unesena slova, odnosno vizualno prikazuje točkice kao znamenke lozinke radi čuvanja privatnosti tijekom unosa podataka. Također, implementirana je i validacija podatka koji se unose u polja što onemogućuje korisniku da nepravilno unosi tražene podatke, primjerice krivo formatirana elektronička pošta kao što je prikazano na slici 4.8. Intuitivnom validacijom korisniku se daje do znanja gdje je krivo unesena neka vrijednost. Neki od predmeta validacije su: krivo formatirana elektronička adresa, slaba lozinka kod registracije, unesen elektronička pošta koja se već koristi u sustavu, netočna lozinka tijekom prijave i sl.



Slika 4.6 Validacija tijekom registracije.



Slika 4.7 Zaslon prijave korisnika.

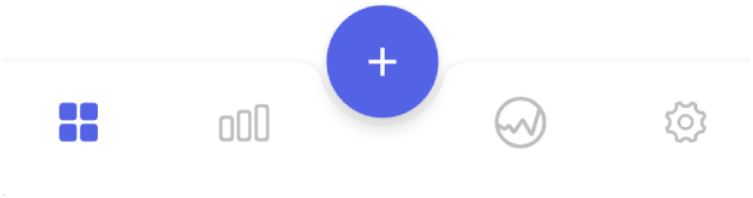


Slika 4.8 Zaslon registracije korisnika.

Od pritiska na akcijsku tipku za prijavu ili registraciju do pojave validacijskog teksta u slučaju neke greške ili do navigacije korisnika na daljnje zaslone pojavljuje se „mrtvo vrijeme“. To je vrijeme tokom kojeg se uneseni podaci procesuiraju na web servisu i šalju nazad kao povratna informacija. Iako se u tom vremenu aktivno događaju određene stvari, korisnik na svom dijelu ne vidi ništa, odnosno aplikacija stoji. Stoga je kao bitan dio korisničkog iskustva važno kroz razne dijelove aplikacije uvesti tzv. indikator napretka u obliku elementa koji daje do znanja korisniku da se stvari učitavaju ili obrađuju. Iz tog razloga tijekom obrade podataka prijave ili registracije, u tom vremenu se na zaslonu prikazuje rotirajući indikator.

4.3. Razvoj glavnih zaslona aplikacije

Nakon uspješne registracije novog korisnika ili prijave već postojećeg, korisnik se prebacuje dalje na glavne zaslone aplikacije kroz koje daljnjom interakcijom koristi aplikaciju. Sučelje aplikacije podijeljeno je na četiri glavna zaslona i jednog posebnog zaslona za unos novih transakcija u sustav, kojima je moguće pristupiti putem navigacijske trake koja se nalazi na samom dnu (slika 4.9).

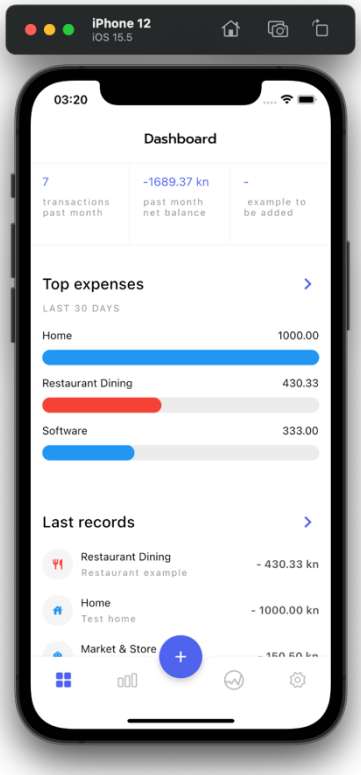


Slika 4.9 Navigacijska traka glavnih zaslona.

Fiksna navigacijska traka ostvarena je putem *BottomAppBar* widgeta koji kao argumente prihvaća ikone pojedinih elemenata koji će se prikazati zajedno s rutama zaslona koji se pristupaju klikom na njih. Pritiskom na pojedinu ikonu korisnik se prebacuje na za to predviđeni zaslon aplikacije.

4.3.1. Početni zaslon

Prvi od četiri glavna zaslona je početni zaslon (slika 4.10). Zadaća ovog zaslona je grubi prikaz određenih podataka kroz statističke segmente čijim se pritiskom korisnika prebacuje na detaljniji prikaz pritisnutog segmenta. Pri vrhu zaslona nalazi se mali prikaz nekih zanimljivih podataka kao što su broj odrađenih transakcija i ukupna razlika prihoda i rashoda za zadnjih mjesec dana. U nastavku zaslona se mogu pronaći statističke kartice koje daju okvirni osvrt na pojedine elemente financija. Pritiskom na njih, otvara se za to namijenjen poseban ekran. Neke od spomenutih kartica su: kartica za prikaz nedavnih transakcija, najvećih troškova, troškova po kategorijama i dr. Cilj ovog zaslona nije dobivanje točnog uvida u stanje financija, već stjecanje brzih i kratkih informacija o trenutnom stanju.

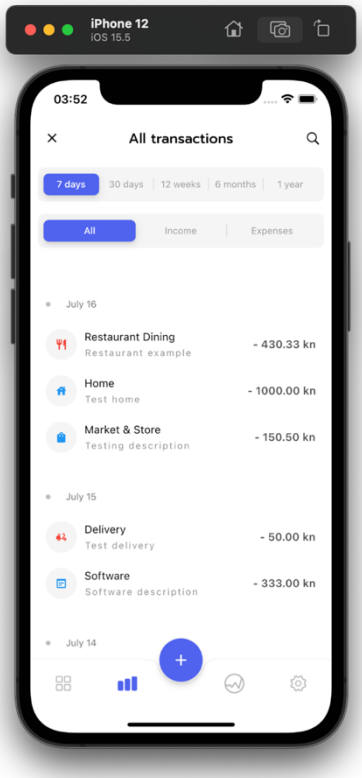


Slika 4.10 Prikaz početnog zaslona.

4.3.2. Zaslon za prikaz transakcija

Sljedeći u redu glavnih zaslona aplikacije je i zaslon namijenjen detaljnom prikazu svih prethodno unesenih transakcija (slika 4.11). Kronološkim redom prikazani su svi unosi uključujući vrstu, kategoriju, iznos, te opis unesenih transakcija. Glavna prednost ovakvog prikaza je jednostavnost pregleda već unesenih transakcija gdje je svaka transakcija grupirana s datumom njenog unosa u sustav. Kako bi taj pregled bio jednostavniji i efikasniji, dostupne su i tri vrste filtra kojima se mogu filtrirati unosi. Jedan od tih filtra je vremenski filtar kojim korisnik može pretraživati transakcije u predefiniranom vremenskom intervalu od zadnjih 7 dana, 30 dana, 12 tjedana, 6 mjeseci ili 1 godinu. Filtar je implementiran s kliznim segmentnim izbornikom čijim je pomicanjem moguće mijenjanje vremenskog intervala dok je početna, zadana opcija vremenskog filtra prikaz zadnjih 7 dana (zbog bržeg učitavanja zaslona s manjim brojem transakcija). Sljedeći filtar, koji je ukomponiran zajedno sa vremenskim filtrom je i filtar prema vrsti transakcije. On omogućuje filtriranje prikaza unesenih transakcija s obzirom na to traže li se transakcije koje

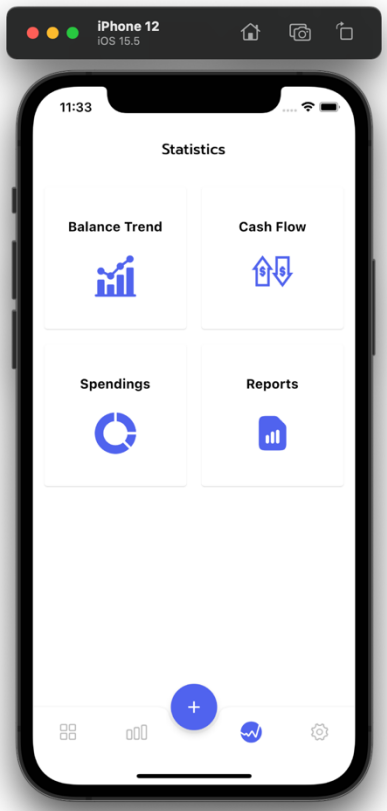
predstavljaju uplatu, isplatu ili oboje. Kao treća mogućnost filtriranja prisutno je i pretraživanje transakcija putem unosa teksta u tekstualno polje pritiskom na ikonu za pretraživanje. Tekst unesen unutar polja pretražuje transakcije preko njihovog opisa koje je korisnik prethodno unio te se vraćaju željeni rezultati. Isto tako, istovremeno se mogu iskombinirati i sve tri navedene filtarske opcije kako bi se omogućio što intuitivniji pregled unesenih transakcija.



Slika 4.11 Zaslon za prikaz transakcija sa aktivnim filtrom.

4.3.3. Zaslon za statistiku

Kao treći od glavnih zaslona, a ujedno bi se moglo reći i najbitniji, je zaslon za prikaz statistike. Na početku je definirano kako je jedna od zadaća ove aplikacije mogućnost praćenja statistike vezane za osobne rashode i prihode. Shodno tome, ovaj zaslon je isključivo namijenjen tome, odnosno sadrži elemente kojima se mogu vizualno i numerički prikazati trenutna financijska stanja prema podacima unesenim od strane korisnika. Prikazan na slici 4.12 je glavni zaslon za prikaz statistike do kojeg se dolazi pritiskom na odgovarajuću ikonu navigacijske trake. Kao što je prikazano, zaslon sadrži četiri interaktivna elementa čijim pritiskom se korisnik vodi na za to odgovarajući posvećen zaslon. Kao četiri glavna moguća statistička pregleda trenutne verzije aplikacije su: ravnotežni trend, protok novca, kategorijalni rashodi i izvještaji.



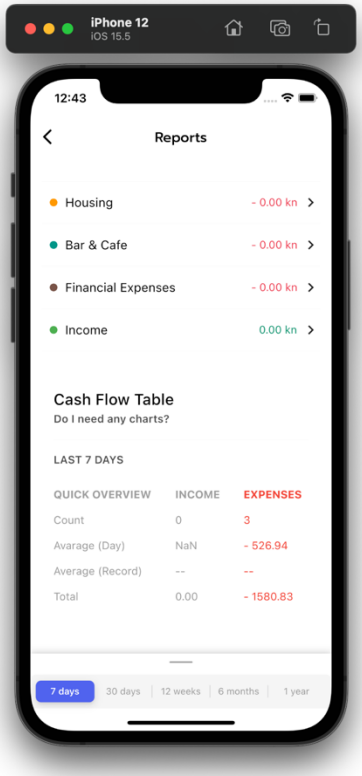
Slika 4.12 Glavni zaslon statistike.

Prvi u nizu pojedinačnih statističkih zaslona je zaslon za prikaz ravnotežnog trenda (slika 4.15). Pomoću njega korisnik može jednostavno, u obliku grafa u koordinatnom sustavu, jednostavno dobiti grafički uvid u omjer ukupne razlike prihoda i rashoda u vremenu.

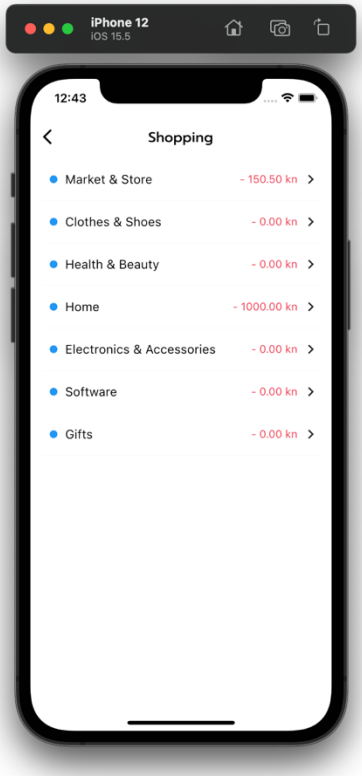
Sljedeći statistički zaslon, prikazan na slici 4.16, daje uvid u ukupni protok novca za određeni vremenski period. Implementirana su dva linearna progresna indikatora koja svojom ispunom prikazuju omjer transakcijskih rashoda i prihoda, te je na početku ispisana ukupna vrijednost razlike prihoda i rashoda u traženom vremenskom periodu.

Pored ova dva navedena statistička prikaza, tu je i prikaz kategorijalnih rashoda (slika 4.17). Prikaz je izveden preko kružnog grafikona koji svojim segmentima prikazuje količinu rashoda na određene kategorije. Isto tako, grafikon je izveden na način da pritiskom na pojedini njegov segment, taj se segment proširuje (time dajući do znanja da je segment aktivan) te se u sredini grafikona ispisuje naziv pritisnute kategorije, zajedno sa iznosom ukupnih rashoda povezanih s tom kategorijom. Segmenti grafikona prikazani su bojama koje odgovaraju kategorijama na koje se odnose.

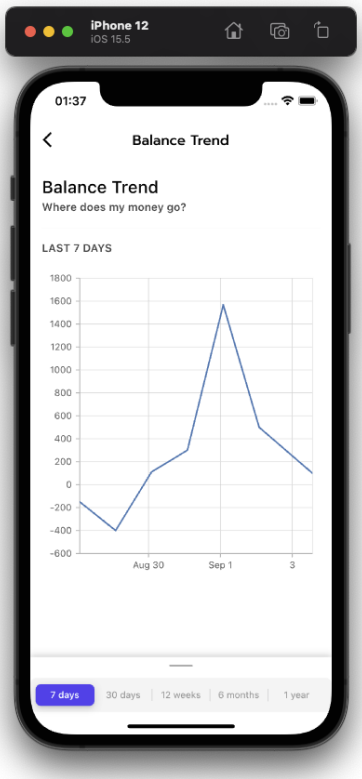
Zadnji u nizu zaslona za statistički pregled je zaslon sa izvještajima (slika 4.18). Može se reći kako je ovaj zaslon, razmatrano u smislu podataka dostupnih za pregled, najdetaljniji. Pomoću njega dostupan je sažeti podatkovni pristup pregledu samih informacija, odnosno bez raznih grafikona, već u čistom obliku prikaza sirovih podataka. Tako je primjerice moguće, kao što je to slučaj kod zaslona kategorijalnih rashoda, vidjeti rashod za pojedinu kategoriju, ali na nešto višoj razini. Dodatni pritiskom na željenu kategoriju, otvara se novi zaslon gdje se ispisuju sve transakcije koje su ostvarene sa odabranom kategorijom zajedno sa pripadajućim datumom, naravno sve u odabranom vremenskom periodu filtera s dna zaslona. Zajedno uz pojedinačni kategorijalni prikaz rashoda, na dnu zaslona se nalazi dodatna tablica koja sadrži određene zanimljive podatke koji bi mogli biti korisni korisniku za znati. Neki od njih su broj pojedinačnih rashodnih i prihodnih transakcija, prosječna razlika prihoda i rashoda po danu, ukupan iznos prihoda ili rashoda te sve to za odabrani vremenski period.



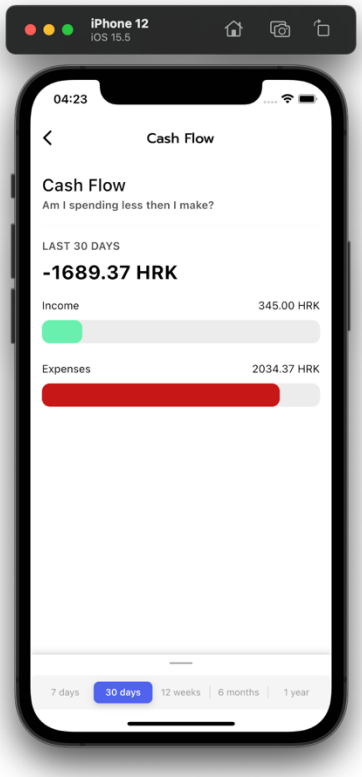
Slika 4.13 Tablica dodatnih informacija izvještaja.



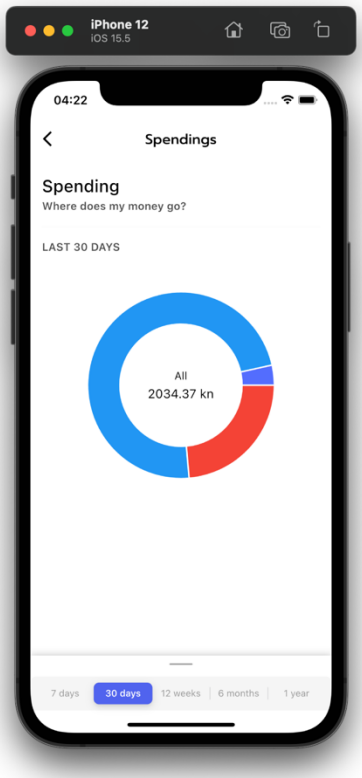
Slika 4.14 Dodatni zaslon prikaza pojedinačnih kategorija.



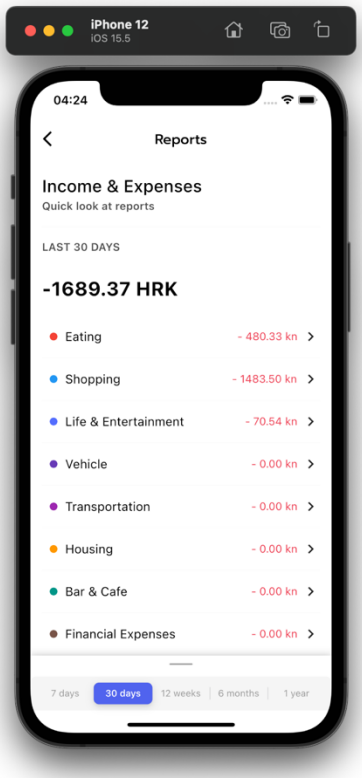
Slika 4.15 Ravnotežni trend.



Slika 4.16 Protok novca.



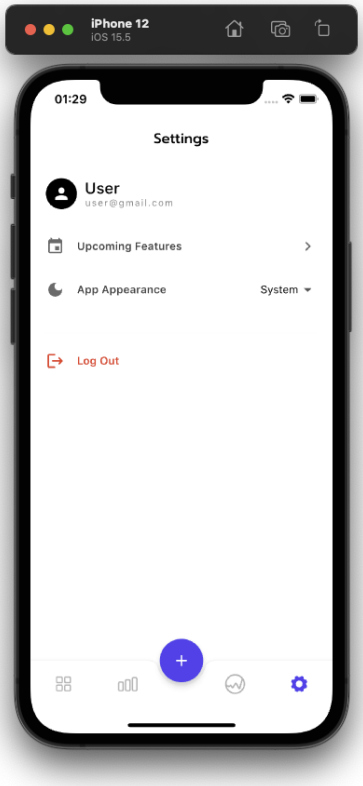
Slika 4.17 Kategorijalni rashodi.



Slika 4.18 Izvještaji.

4.3.4. Zaslon postavki

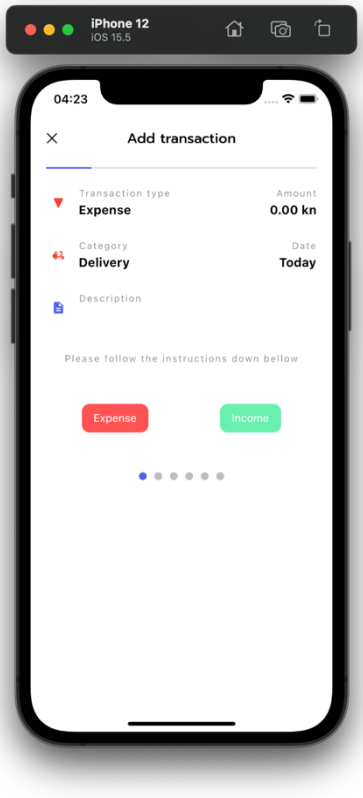
Zaslon postavki (slika 4.19) sadrži par elemenata, ne nužno bitnih za sam rad aplikacije, nego više kao popratne funkcije i sadržaj. Na vrhu samog zaslona mogu se pronaći podaci od trenutno prijavljenom korisniku, odnosno njegovo korisničko ime unijeto tijekom registracije, te elektronička pošta koja se koristi za prijavu u aplikaciju. U nastavku se može otvoriti i lista nadolazećih značajki u sustav rada aplikacije, odnosno buduće nove implementacije koje se mogu očekivati. Nadalje, značajka koja se nije spominjala do sada, a moguće ju je mijenjati izravno s ovog zaslona, je opcija promjene teme aplikacije. Implementirane su tri moguće opcije odabira teme: tamni način, svijetli način i opcija praćenja teme koja je opće aktivna na sustavu uređaja koji pokreće aplikaciju (ujedno i zadana opcija). Pored navedenih opcija, na samom dnu se nalazi i tipka kojom se korisnik odjavljuje iz sustava te se prebacuje na zaslon za prijavu ili registraciju gdje se može ponovno prijaviti s istim ili drugim računom.



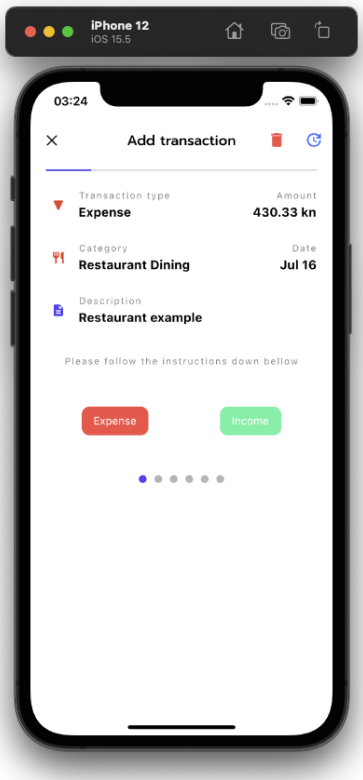
Slika 4.19 Zaslon postavki.

4.3.5. Zaslon za dodavanje novih transakcija

Kako bi uopće bilo moguće dobiti generalni uvid u osobne financije putem ove aplikacije, potrebno je omogućiti korisniku unos samih transakcija koje je izvršio. Za tu posebnu namjenu izveden je zaslon za dodavanje novih transakcija (slika 4.20). Pritiskom na središnju tipku u navigacijskoj traci, otvara se zaslon kao što je prikazano na slici. Postupak unosa nove transakcije vrši se u šest koraka, a informacija u izvršnim koracima korisniku je prikazana u obliku linearnog progresnog indikatora na vrhu i točkastog, koračnog indikatora na dnu. Praćenjem traženih uputa na dnu i ispunjavanjem tekstualnih i brojčanih polja (slike 4.22 - 4.27), popunjavaju se elementi iz gornjeg dijela zaslona (vrsta transakcije, iznos, kategorija, datum i opis). Važno je napomenuti kako su ovi prikazani elementi ujedno i interaktivni, odnosno pritiskom na element, moguće je preskočiti korak ili vratiti se na prethodni. Time se omogućuje ne skroz linearni proces, već i mogućnost povratka na pojedine dijelove procesa unosa i ispravljanje prema potrebi. Nakon što se popune sva tražena polja i pritisne tipka za spremanje, podaci se obrađuju te ako zadovoljavaju kriterij se spremaju u bazu podataka.



Slika 4.20 Zaslon za unos transakcija.

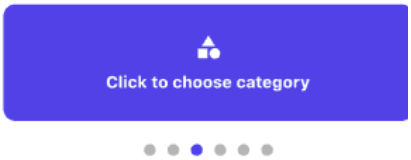


Slika 4.21 Zaslon za izmjenu ili brisanje transakcije.

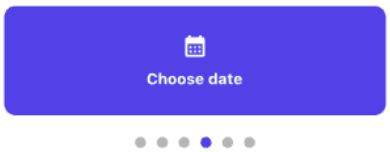
Zaslonu za unos transakcija je moguće pristupiti i na još jedan način. Na zaslonu prikaza svih transakcija, pritiskom na pojedinu transakciju korisniku se omogućuje izmjena ili brisanje već postojeće transakcije te se otvara zaslon isti ovome, izuzev dvije nove tipke pri vrhu (slika 4.21). Korisnik može izmijeniti bilo koji aspekt već postojeće transakcije i pritiskom na gornju desnu tipku izvršiti unesenu promjenu. Isto tako, transakciju je moguće i potpuno izbrisati pritiskom gornje lijeve tipke za brisanje čime se odabrana transakcija u potpunosti briše iz sustava.

Slika 4.22 Odabir vrste transakcije.

Slika 4.23 Unos iznosa transakcije.

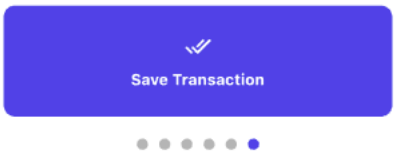


Slika 4.24 Odabir kategorije.



Slika 4.25 Odabir datuma.

Slika 4.26 Unos opisa transakcije.



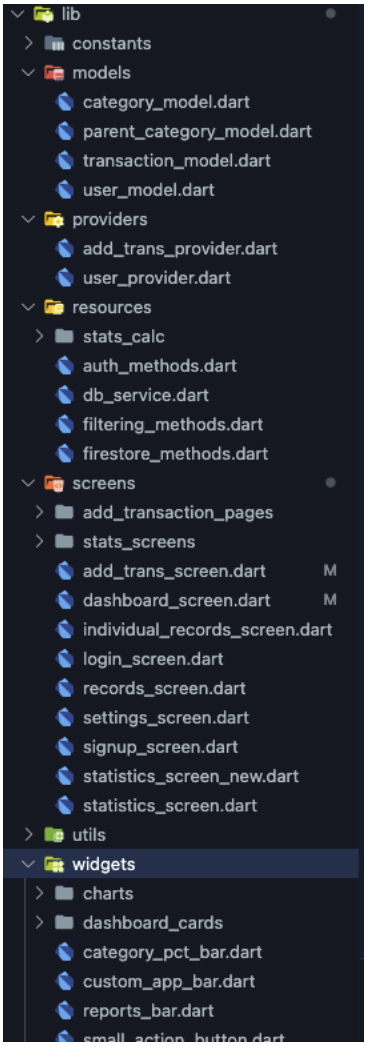
Slika 4.27 Potvrda unosa.

4.4. Proces strukturiranja radnih datoteka i modularnost

Do sad je u radu predstavljena aplikacija kroz oči korisnika, odnosno kroz korisničko sučelje. Međutim, ako se pogleda u pozadinu svega toga, iza sučelja same aplikacije nalazi se velika količina programskog koda koja je zadužena za sve što korisnik vidi i ne vidi tijekom korištenja aplikacije na svojem uređaju. Kako projekt (program) raste, tako se i količina koda i raznih drugih dodataka (slike, fontovi, ikone i sl.) sukladno povećava. Da bi efikasnost procesa razvoja ostala netaknuta, važno je držati se nekih osnovnih preporuka prilikom razvoja. Najbitnija ideja, kojom je vođen svaki dobar i uspješni projekt, je održavanje modularnosti projekta. Na ovom konkretnom primjeru razvoja mobilne aplikacije ta praksa modularnosti je izvedena na način da se glavna baza koda dijeli na puno manjih zasebnih dijelova koji zajedno čine jednu cjelinu. Tako su primjerice određeni elementi koji se pojavljuju na zaslonima kodno izolirani, odnosno kod kojim su napisani je izveden u zasebnoj datoteci spremljenoj u radnoj mapi. Tom praksom su ostvarene neke od važnih značajki kao što su jednostavnost modifikacije, kodna sigurnost i ponovno korištenje istog koda. Principi jednostavnosti modifikacije i ponovnog korištenja istog koda idu jedan uz drugi. Drugim riječima, ako u programu postoji neki dio koji se često ponavlja, dobra je ideja izolirati ga u samostalnu datoteku te ga iz te vanjske datoteke pozivati u glavni kod po potrebi (slično namijeni funkcija ili metoda u generalnom programiranju). Isto tako, ako dođe do potrebe izmjene određenog aspekta programa te ukoliko su primijenjena ova načela, taj zahtjev

za modifikaciju puno će se lakše izvesti na način da će biti potrebna promjena programskog koda samo na jednom mjestu u projektu. Time se znatno smanjuju vjerojatnosti neusklađenosti i pogrešaka te se skraćuje vrijeme koje je potrebno uložiti u traženu doradu, odnosno modifikaciju. Ovaj princip se često zanemaruje u početku rada na projektu (uglavnom kod novijih razvojnih programera) iz razloga što je u početku lakše sve „ugurati u jedno“ naspram primjene principa modularnosti. U početku to nije toliki problem, međutim, kako se projekt širi i razvija, efekt nemodularnosti dolazi do izražaja i daljnje samo otežava razvojni proces za sve sudionike projekta. Kodna sigurnost, kao još jedna dobra značajka modularnosti može u velikoj mjeri smanjiti vrijeme potrebno za pronalazak eventualne greške u kodu koja se može dogoditi. Klasični primjer pisanja sigurnog koda je odvajanje logičkog dijela programa (dio zadužen za proračune i općenite stvari vezane za *backend*[5]) od onog prezentacijskog koji krajnji korisnik koristi.

Na slici 4.28 prikazan jedan dio strukture radne mape u kojoj se razvija aplikacija ovog rada. Ukupan broj radnih datoteka u ovom projektu prelazi brojku od 50 (nisu sve prikazane na slici), te je vidljivo kako je nužno imati neku vrstu organizacije svih tih datoteka da bi razvoj mogao biti efikasan. U ovom projektu, odlučeno je strukturirati radne datoteke po smislenim cjelinama, odnosno mapama te njihovim podmapama. Tako su neke od mapa one za zaslone, modele, resurse, ponavljajuće widgete, konstante aplikacije i dr. Ovakvom podjelom ostvarena je mogućnost jednostavnijeg snalaženja tijekom aktivnog razvoja što uvelike smanjuje vrijeme potrebno za razvoj same aplikacije ili programa.



Slika 4.28 Dio strukture radne mape.

[5] Engl. Backend - Dio računalnog sustava ili programa kojem korisnik ne pristupa izravno, uglavnom odgovoran za pohranu i obradu podataka

5. ZAKLJUČAK

Današnjim ubrzanim tijekom života lako se može zapostaviti svijest o vlastitim financijama, bilo to radi kompleksnosti načina vođenja same evidencije ili jednostavnog nemara i nezainteresiranosti. Međutim, ono ne mora nužno biti teško. Ovim završnim radom prikazano jedno od rješenja, upravljanje osobnim financijama putem osobne mobilne aplikacije. Osmišljeno i izvedeno tako da se korisnik sa svojim računom prijavljuje u aplikaciju u kojoj mu je omogućeno jednostavno i brzo unošenje dnevnih transakcija, pregled istih te uvid u statistiku prihoda i rashoda. Prednost ovakvog vođenja osobnih financija je efikasnost, što znači da ne zahtijeva puno pažnje i uloženog truda od same osobe kao što bi primjerice trebalo kod klasičnog vođenja financija putem unosa prihoda i rashoda u knjige.

6. LITERATURA

[1] „Kućni budžet“, https://specijali.net.hr/financije/pratite-li-svoj-kucni-budzet, s interneta, pristupljeno 15.05.2022

[2] „Mobile operating systems“, https://www.sciencedirect.com/topics/computer-science/mobile-operating-system, s interneta, pristupljeno 21.05.2022

[3] „The Difference Between Native And Cross-Platform Development“, https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019 , s interneta, pristupljeno 12.06.2022

[4] „What is the Flutter framework“, https://www.perfecto.io/blog/what-is-flutter-framework, s interneta, pristupljeno 12.06.2022

[5] „What is version control“, https://www.atlassian.com/git/tutorials/what-is-version-control, s interneta, pristupljeno 23.06.2022

[6] „Brand Guidlines“, https://source.android.com/setup/start/brands , s interneta, pristupljeno 15.05.2022

[7] „Design“, https://developer.apple.com/design/human-interface-guidelines/foundations/app-icons, s interneta, pristupljeno 15.05.2022

[8] „Brand“, https://flutter.dev/brand , s interneta, pristupljeno 15.05.2022

[9] J. Andress, *The Basics of Information Security (Second Edition)*, 2014

[10] Hamed T, Kremer S, *Computer and Information Security Handbook (Third Edition)*, 2017

7. DODACI

Popis slika

Popis tablica

8. PRILOG 1: SAŽETAK GLAVNOG KODA

Datoteka **main.dart** - Glavna, ujedno i početna datoteka programa od koje sve počinje

```
import 'package:adaptive_theme/adaptive_theme.dart';
import 'package:finance_tracker/constants/colors.dart';
import 'package:finance_tracker/providers/add_trans_provider.dart';
import 'package:finance_tracker/providers/user_provider.dart';
import 'package:finance_tracker/screens/login_screen.dart';
import 'package:finance_tracker/themes.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:provider/provider.dart';
import 'bottom_navigation.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return ScreenUtilInit(
      designSize: const Size(360, 640),
      builder: (context, child) => MultiProvider(
        providers: [
          ChangeNotifierProvider(create: (_) => UserProvider()),
          ChangeNotifierProvider(create: (_) => AddTransProvider()),
        ],
        child: AdaptiveTheme(
          light: lightModeTheme(),
          dark: darkModeTheme(),
          initial: AdaptiveThemeMode.system,
          builder: (theme, darkTheme) {
            return MaterialApp(
              debugShowCheckedModeBanner: false,
              title: 'Finance Tracker',
              theme: theme,
              darkTheme: darkTheme,
              home: StreamBuilder(
                stream: FirebaseAuth.instance.authStateChanges(),
```

```
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.active) {
                if (snapshot.hasData) {
                  return const BottomNavigation();
                } else if (snapshot.hasError) {
                  return Center(
                    child: Text('${snapshot.error}'),
                  );
                }
              }
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const Center(
                  child: CircularProgressIndicator(
                    color: primaryColor,
                  ),
                );
              }

              return const LoginScreen();
            },
          ),
        );
      },
    ),
  ),
);
  }
}
```

Datoteka **dashboard_screen.dart** - Datoteka početnog zaslona aplikacije

```
import 'package:finance_tracker/resources/filtering_methods.dart';
import 'package:finance_tracker/resources/stats_calc/cash_flow_calc.dart';
import 'package:finance_tracker/resources/stats_calc/reports_calc.dart';
import 'package:finance_tracker/screens/stats_screens/spendings_screen.dart';
import 'package:finance_tracker/widgets/custom_app_bar.dart';
import 'package:finance_tracker/widgets/dashboard_cards/expenses_card.dart';
import 'package:finance_tracker/widgets/dashboard_cards/last_records_card.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:provider/provider.dart';

import '../models/transaction_model.dart';
import '../models/user_model.dart';
import '../providers/user_provider.dart';
import '../resources/db_service.dart';
import '../widgets/dashboard_cards/top_expenses_card.dart';

class DashboardScreen extends StatefulWidget {
  const DashboardScreen({Key? key}) : super(key: key);

  @override
  State<DashboardScreen> createState() => _DashboardScreenState();
}

class _DashboardScreenState extends State<DashboardScreen> {
  // Filtering Services
  FilteringMethods filtering = FilteringMethods();

  // Reports Services
  ReportsCalc reports = ReportsCalc();

  // CashFlow calc
  CashFlowCalc cashFlowCalc = CashFlowCalc();

  @override
  Widget build(BuildContext context) {
    UserModel? user = Provider.of<UserProvider>(context).getUser;

    return Scaffold(
      appBar: CustomAppBar(
        title: 'Dashboard',
      ),
      body: SingleChildScrollView(
          child: StreamBuilder<List<TransactionModel>>(
        stream: DatabaseServices().getTransactionsStream(user!),
        builder: (context, snapshot) {
          if (!snapshot.hasData) {
            return const Center(
              child: Text('NO TRANSACTIONSs'),
            );
          }
```

```
final List<TransactionModel> transactions =
    filtering.getFilteredTrans2(
  snapshot.data!,
  1,
  0,
);

return Column(
  children: [
    Container(
      decoration: BoxDecoration(
        border: Border(
          top: BorderSide(
              width: 1, color: Theme.of(context).dividerColor),
          bottom: BorderSide(
              width: 1, color: Theme.of(context).dividerColor),
        ),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          ColumnItem(
              titleValue:
                  reports.transactionCount(transactions, true) +
                      reports.transactionCount(transactions, false),
              subtitle: 'transactions past month'),
          ColumnItem(
              titleValue:
                  '${cashFlowCalc.cashFlowTotalCalc(transactions).toStringAsFixed(2)} kn',
              subtitle: 'past month net balance'),
          ColumnItem(
              titleValue:
                  '${reports.getLargestrTransaction(transactions).toStringAsFixed(2)} kn',
              subtitle: 'largest transaction'),
        ],
      ),
    ),
    const SizedBox(height: 20),
    InkWell(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => const SpendingsScreen()),
        );
      },
      child: TopExpensesCard(transactions),
    ),
    const SizedBox(height: 20),
    InkWell(
      onTap: () {
        Navigator.push(
```

```
                context,
                MaterialPageRoute(
                    builder: (context) => const SpendingsScreen()),
              );
            },
            child: LastRecordsCard(),
          ),
          const SizedBox(height: 20),
          InkWell(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => const SpendingsScreen()),
              );
            },
            child: ExpensesCard(transactions),
          ),
        ],
      );
    },
  )),
 );
 }
}

class ColumnItem extends StatelessWidget {
 final titleValue;
 final subtitle;

 const ColumnItem({Key? key, required this.titleValue, required this.subtitle})
    : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Container(
   height: 80.h,
   decoration: BoxDecoration(
    border: Border(
     //top: BorderSide(width: 1, color: Color(0xffe0dfe7)),
     //bottom: BorderSide(width: 1, color: Color(0xffe0dfe7)),
     right: BorderSide(width: 1, color: Theme.of(context).dividerColor),
    ),
   ),
   padding: const EdgeInsets.all(15),
   width: MediaQuery.of(context).size.width * 0.33,
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Text(
      titleValue.toString(),
      style: TextStyle(
       fontSize: 14.sp,
```

```
            color: Theme.of(context).primaryColor,
          ),
        ),
        const SizedBox(height: 10),
        Text(
          subtitle,
          softWrap: true,
          style: Theme.of(context).textTheme.labelSmall,
        ),
      ],
    ),
  );
 }
}
```

Datoteka **statistics_screen.dart** - Datoteka glavnog prikaza statističkih jedinica

```
import 'package:finance_tracker/screens/stats_screens/balance_trend_screen.dart';
import 'package:finance_tracker/screens/stats_screens/cash_flow_screen.dart';
import 'package:finance_tracker/screens/stats_screens/reports_screen.dart';
import 'package:finance_tracker/screens/stats_screens/spendings_screen.dart';
import 'package:finance_tracker/widgets/custom_app_bar.dart';
import 'package:flutter/material.dart';

class StatisticsScreenNew extends StatelessWidget {
 const StatisticsScreenNew({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: CustomAppBar(
    title: 'Statistics',
   ),
   body: GridView.count(
    padding: const EdgeInsets.all(10),
    crossAxisSpacing: 10,
    mainAxisSpacing: 10,
    crossAxisCount: 2,
    children: [
     GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => const BalanceTrendScreen()),
       );
      },
      child: const StatsCard(
          title: 'Balance Trend', image: 'assets/images/statistics.png'),
     ),
     GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const CashFlowScreen()),
       );
      },
      child: const StatsCard(
          title: 'Cash Flow', image: 'assets/images/income.png'),
     ),
     GestureDetector(
      onTap: () {
       Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => const SpendingsScreen()),
       );
      },
```

```
            child: const StatsCard(
                title: 'Spendings', image: 'assets/images/pie-chart-2.png'),
          ),
          GestureDetector(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => const ReportsScreen()),
              );
            },
            child: const StatsCard(
                title: 'Reports', image: 'assets/images/reports.png'),
          ),
        ],
      ),
    );
  }
}

class StatsCard extends StatelessWidget {
  final String title;
  final String image;

  const StatsCard({Key? key, required this.title, required this.image})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              title,
              style: Theme.of(context).textTheme.labelMedium,
            ),
            const SizedBox(height: 25),
            SizedBox(
              height: 50,
              width: 50,
              child: Image.asset(image),
            ),
          ],
        ),
      ),
    );
  }
}
```

9. SAŽETAK I KLJUČNE RIJEČI

Kroz ovaj završni rad, obrađena je tematika mobilnih operacijskih sustava te njihovih generalnih podjela. Objašnjeni su pojmovi nativnog i višeplatformskog razvoja aplikacija, njihove generalne razlike te prednosti i mane. Nadalje, kao glavna tematika ovog završnog rada prikazan je generalni postupak razvoja aplikacije za praćenje osobnih financija od postavka alata potrebnih za rad do razvoja samih pojedinačnih zaslona. Za kraj, prikazan je proces strukturiranja radnih datoteka te je objašnjena važnost primjene principa modularnosti tijekom razvoja većeg projekta.

Ključne riječi: Flutter, višeplatformska aplikacija, financije, programiranje

10. SUMMARY AND KEY WORDS

Throughout this bachelor thesis, the topic of mobile operating systems and their general categorizations is covered. The concepts of native and cross-platform application development, their general differences and the advantages and disadvantages are explained. Furthermore, as the main topic of this bachelor thesis, the general process of application development for monitoring personal finances, from setting up the necessary tools for work to the development of the individual screens, is presented. Finally, the process of structuring work files is presented and the importance of applying the principle of modularity during development of a larger project is explained.

Key words: Flutter, cross-platform application, finances, programming,