# Android aplikacija za praćenje kućnog budžeta

**Čaušić, Benjamin**



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

**Sveučilišni diplomski studij Računarstvo**

# ANDROID APLIKACIJA ZA PRAĆENJE KUĆNOG BUDŽETA

**Diplomski rad**

**Benjamin Čaušić**

**Osijek, 2024.**

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

**Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju**

## Ocjena diplomskog rada na sveučilišnom diplomskom studiju

| | |
|---|---|
| **Ime i prezime pristupnika:** | Benjamin Čaušić |
| **Studij, smjer:** | Sveučilišni diplomski studij Računarstvo |
| **Mat. br. pristupnika, god.** | D1275R, 07.10.2022. |
| **JMBAG:** | 0165071076 |
| **Mentor:** | doc. dr. sc. Tomislav Galba |
| **Sumentor:** | |
| **Sumentor iz tvrtke:** | |
| **Predsjednik Povjerenstva:** | izv. prof. dr. sc. Časlav Livada |
| **Član Povjerenstva 1:** | doc. dr. sc. Tomislav Galba |
| **Član Povjerenstva 2:** | izv. prof. dr. sc. Alfonzo Baumgartner |
| **Naslov diplomskog rada:** | Android aplikacija za praćenje kućnog budžeta |
| **Znanstvena grana diplomskog rada:** | **Programsko inženjerstvo (zn. polje računarstvo)** |
| **Zadatak diplomskog rada:** | Potrebno je napraviti mobilnu aplikaciju za Android mobilne uređaje koristeći neku od postojećih tehnologija (Kotlin/Java). Aplikacija treba omogućiti registraciju korisnika te unos i pregled prihoda i rashoda (troškovi kućanstva (hrana,piće,režije), dodatni troškovi). S ciljem olakšavanja korisnicima unos podataka po mogućnosti implementirati automatsko prepoznavanje računa uz pomoć kamere (korištenje OCR-a, biblioteka za manipulaciju QR kodova). Napomena: tema rezervirana za Benjamin Čaušić |

| | |
|---|---|
| **Datum ocjene pismenog dijela diplomskog rada od strane mentora:** | 18.09.2024. |
| **Ocjena pismenog dijela diplomskog rada od strane mentora:** | Izvrstan (5) |
| **Datum obrane diplomskog rada:** | 27.09.2024. |
| **Ocjena usmenog dijela diplomskog rada (obrane):** | Izvrstan (5) |
| **Ukupna ocjena diplomskog rada:** | Izvrstan (5) |
| **Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:** | 27.09.2024. |

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

## IZJAVA O IZVORNOSTI RADA

**Osijek,** 27.09.2024.

| | |
|---|---|
| **Ime i prezime Pristupnika:** | Benjamin Čaušić |
| **Studij**: | Sveučilišni diplomski studij Računarstvo |
| **Mat. br. Pristupnika, godina upisa:** | D1275R, 07.10.2022. |
| **Turnitin podudaranje [%]:** | 9 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za praćenje kućnog budžeta**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

**SADRŽAJ**

# 1. UVOD

U današnjem ubrzanom svijetu praćenje osobnih financija postaje sve izazovnije. Puno ljudi ima poteškoće s bilježenjem svojih prihoda i rashoda što može dovesti do lošeg upravljanja financijama i nepotrebnih dugova. Zahvaljujući širokoj dostupnosti pametnih telefona mobilne aplikacije predstavljaju praktičan alat za rješavanje ovog problema.

Ovaj se diplomski rad bavi razvojem mobilne aplikacije za Android uređaje koja omogućuje korisnicima jednostavno i efikasno upravljanje svojim financijama. Aplikacija je razvijena koristeći Kotlin/Java tehnologiju, s naglaskom na korisničko iskustvo i funkcionalnost. Ključne značajke aplikacije uključuju registraciju korisnika, unos i pregled prihoda i rashoda te kategorizaciju troškova u kućanstvu (hrana, piće, režije) i dodatnih troškova.

Kako bi se korisnicima dodatno olakšao unos podataka, aplikacija će biti opremljena funkcionalnošću automatskog prepoznavanja računa uz pomoć kamere. Ova funkcionalnost koristi tehnologiju optičkog prepoznavanja znakova OCR (*Optical character recognirion*) i biblioteke za manipulaciju QR (*Quick Response Code*) *kôdovima*. Automatsko prepoznavanje računa omogućuje brzo i precizno bilježenje troškova smanjujući potrebu za ručnim unosom podataka.

Cilj je ovog rada istražiti i implementirati navedene funkcionalnosti te evaluirati učinkovitost aplikacije u kontekstu olakšavanja upravljanja osobnim financijama. Razvoj aplikacije uključivat će detaljnu analizu potreba korisnika, dizajn i implementaciju sustava te testiranje i evaluaciju konačnog proizvoda. Očekuje se da će ovakva aplikacija doprinijeti boljem financijskom planiranju i kontroliranju troškova pružajući korisnicima alat za efikasnije upravljanje njihovim financijama.

U drugom poglavlju objašnjavaju se osnovni koncepti i teorije relevantne za razvoj mobilnih aplikacija i upravljanje osobnim financijama. Treće poglavlje analizira trenutna rješenja za upravljanje osobnim financijama putem mobilnih aplikacija, s naglaskom na prednosti i nedostatke postojećih aplikacija. Četvrto poglavlje donosi detaljan opis alata i tehnologija korištenih za izradu aplikacije, uključujući razvojna okruženja, programske jezike i biblioteke. Peto poglavlje pruža detaljan uvid u organizaciju i dizajn aplikacije unutar Android Studio razvojnog okruženja, s posebnim fokusom na implementaciju modularnog pristupa, mogućnosti skaliranja te osiguravanje održivosti aplikacije kroz vrijeme. Šesto poglavlje prikazuje korisničko sučelje i ključne funkcionalnosti aplikacije uključujući primjere

*kôda* koji omogućuju ispravno funkcioniranje aplikacije.

# 2. ULOGA MOBILNIH APLIKACIJA U ŠTEDNJI

Mobilne aplikacije imaju ključnu ulogu u upravljanju osobnim financijama, a poseban se naglasak stavlja na štednju. Korištenjem aplikacija za osobne financije korisnici mogu pratiti svoje troškove i planirati štednju putem preglednika ili preuzimanjem aplikacija na pametne telefone i tablete. U razvijanju takvih aplikacija uvijek se pazi na jednostavnost korištenja tako da su sučelja razvijena s ciljem povećanja praktičnosti tako da prosječan korisnik nema većih problema u korištenju istih. Također, važno je istaknuti pristupačnost ovih aplikacija koje korisnicima omogućuju pristup financijskim podacima u bilo kojem trenutku i na bilo kojem mjestu. Time se štedi vrijeme te olakšava donošenje informiranih odluka o štednji. Prilikom korištenja aplikacija nije potrebna nikakva tehnička ili financijska stručnost [1].

French, McKillop i Stewart (2020) istraživali su učinkovitost osobnih financijskih aplikacija za pametne telefone u poboljšanju financijske sposobnosti osoba s niskim primanjima. Razvili su četiri aplikacije za pametne telefone (aplikacija za usporedbu kamatnih stopa na zajmove, aplikacija za usporedbu troškova, aplikacija za gotovinski kalendar i aplikacija za upravljanje dugom). Ove aplikacije bile su dostupne članovima Derry Credit Uniona, najveće kreditne unije u Sjevernoj Irskoj. Članovi kreditnih unija u Sjevernoj Irskoj obično imaju relativno niže prihode u usporedbi s općom populacijom i žive u socijalno-ekonomski nepovoljnijim područjima. Nasumično kontrolirano ispitivanje RCT (*Randomized Controlled Trial*) korišteno je za procjenu učinkovitosti aplikacija u poboljšanju financijski prihvatljivog ponašanja. Poboljšanja su primijećena u nekoliko segmenata; procjeni financijskog znanja, razumijevanju i osnovnim vještinama te u stavovima i motivaciji [2].

## 2.1. Mobilne aplikacije – prednosti i izazovi

Mobilne aplikacije, definirane kao tehnologija koja se preuzima i koristi na pametnim telefonima i tabletima, postale su neizostavan dio suvremenog života. Njihova upotreba raste globalno, osobito u specifičnim industrijama poput bankarstva, zdravstva, obrazovanja i marketinga. Osim toga, mobilne aplikacije pronalaze široku primjenu u drugim sektorima. Koriste se za rezervacije hotela, zdravstvenu skrb, visoko obrazovanje, sport i kućne potrebe. Povezivanje mobilnih aplikacija s poslovnim procesima smanjuje vrijeme odaziva korisnika i povećava učinkovitost radne snage.

Međutim, postoje i izazovi u korištenju mobilnih aplikacija. Na primjer, u visokom

obrazovanju, izazovi uključuju nedostatak propusnosti, sigurnosne sustave, obučeno osoblje i financijska sredstva. Nadalje, u zdravstvenom sektoru, problemi su loša kvaliteta aplikacija te nedostatak društvenog umrežavanja.

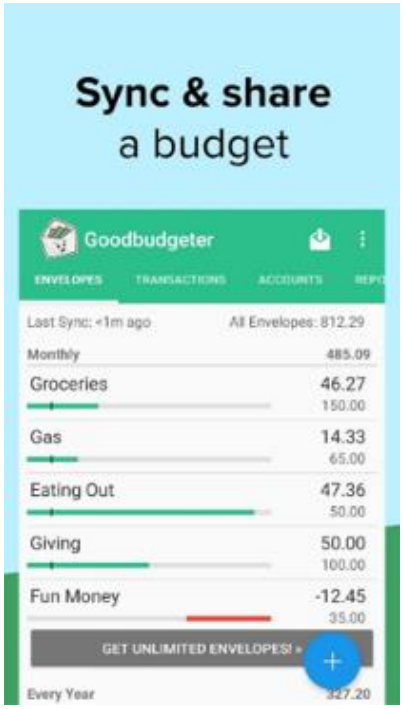
Studije o kućnoj štednji i potrošnji obuhvaćaju različite aspekte, uključujući mjesečne troškove za hranu, elastičnost troškova hrane i potražnju za specifičnim namirnicama. Poveznice između izdvajanja budžeta i potražnje za glavnim stavkama potrošnje kao što su tekstil, namirnice, osobna njega i zdravstvena skrb, sve su veći predmet interesa [3].

## 2.2. Praćenje financijskih ciljeva s Android aplikacijama

Korištenje aplikacija za pametne telefone s utjecajem na osobne financije rezultira poboljšanjem različitih mjera koje se koriste za procjenu financijskih znanja i vještina, stavova i motivacije te financijski prihvatljivih ponašanja za one u kućanstvima s niskim prihodima. Osobe koje su imale aplikacije za praćenje troškova pokazale su povećano samopouzdanje u donošenju financijskih odluka, kao i poboljšanu financijsku pismenost. Financijski prihvatljiva ponašanja očituju se u boljoj sposobnosti praćenja financija i upravljanjem neočekivanih troškova [2].

## 2.3. Pregled aktualnih aplikacija

Na slici 2.1. prikazan je Goodbudget, napredni alat za upravljanje novcem i praćenje troškova, koji je posebno pogodan za planiranje kućnog proračuna. Ovaj osobni financijski menadžer modernizira tradicionalni sustav omotnica, omogućujući proaktivni plan proračuna koji pomaže korisnicima da prate račune i financije. Dizajniran za jednostavno i pravovremeno praćenje, omogućuje sinkronizaciju i dijeljenje podataka putem Android i iPhone uređaja, kao i weba, čime osigurava da svi članovi kućanstva budu informirani o financijskom stanju domaćinstva. Osim intuitivnog sučelja, Goodbudget nudi napredne tehnologije za praćenje i analizu, čineći ga nezaobilaznim alatom za učinkovito upravljanje financijama [4].




***Slika 2.1.*** *Sučelje Goodbudget aplikacije [4]*

Na slici 2.2. prikazana je aplikacija YNAB (*You Need A Budget*) koja korisnicima omogućuje upravljanje svojim financijama s povjerenjem i jasnoćom. Pomoću YNAB metode korisnici mogu planirati i pratiti svoje troškove i štednju putem jednostavnih, ali učinkovitih pravila koja obuhvaćaju davanje svakom dolaru određene uloge, prilagodbu stvarnim troškovima, fleksibilno prilagođavanje promjenama te korištenje prošlogodišnje zarade za tekuće troškove. YNAB je dostupan na svim uređajima, uključujući web, iPhone, Android, iPad i Apple Watch [5].


***Slika 2.2.*** *Sučelje YNAB aplikacije [6]*

# 3. OPIS PRIMJENJENIH ALATA I TEHNOLOGIJA

Treće poglavlje pružit će detaljan pregled i temeljitu analizu tehnologija koje su korištene u procesu razvoja aplikacije ističući njihovu važnost i ulogu u postizanju funkcionalnosti sustava.

## 3.1. Android Studio

Android Studio predstavlja službeno integrirano razvojno okruženje IDE (*Integrated Development Environment)* za razvoj aplikacija na Android platformi. Omogućava programerima stvaranje aplikacija visoke kvalitete. Temeljen na IntelliJ IDEA platformi, jednom od vodećih razvojnih okruženja za Javu, Android Studio sadrži sve potrebne alate i funkcionalnosti za cjelokupni proces razvoja Android aplikacija. Na slici 3.1. prikazan je pregled sučelja Android Studija koji omogućuje sve korake razvoja aplikacije – od pisanja *kôda* preko testiranja pa sve do implementacije. Android Studio pruža cjelovite funkcionalnosti koje zadovoljavaju potrebe kako početnika, tako i iskusnih programera. Ovaj alat pruža sve što je potrebno za stvaranje vrhunskih Android aplikacija omogućujući učinkovit i efikasan rad kroz cijeli razvojni ciklus [7].

Softver je premijerno prikazan na Google I/O konferenciji u svibnju 2013., a stabilna verzija postala je dostupna krajem 2014. godine. Android Studio podržava rad na operativnim sustavima macOS, Windows i Linux te je zamijenio Eclipse Android Development Tools (ADT) kao glavno razvojno okruženje za izradu aplikacija namijenjenih Android platformi.

Android Studio koristi Gradle sustav za izgradnju, Android emulator, gotove predloške *kôda* i GitHub integraciju kako bi olakšao razvoj aplikacija za Android operativni sustav. Svaki projekt unutar Android Studija sastoji se od jednog ili više modaliteta koji uključuju izvorni *kôd* i pripadajuće resurse. Ovi modaliteti uključuju module kao što su Android aplikacija, module knjižnica i module Google App Engine.

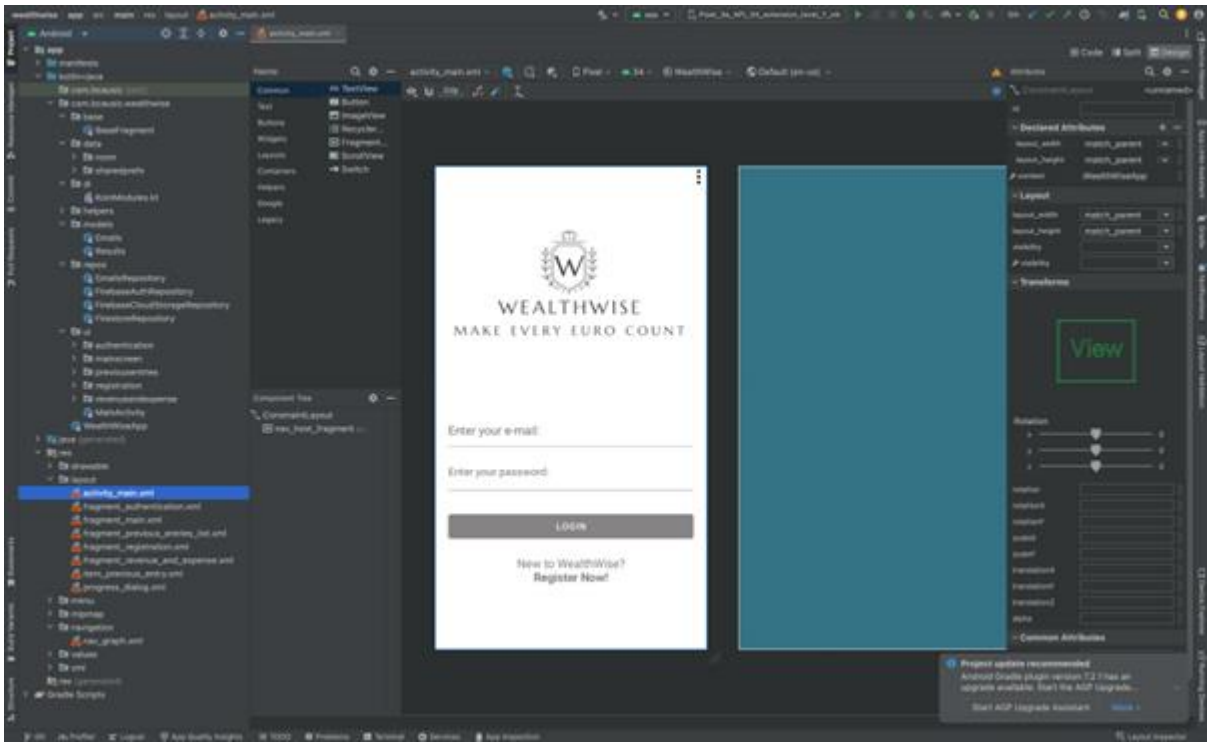
Uređivač *kôda* unutar Android Studija pomaže programerima prilikom pisanja *kôda* nudeći funkcionalnosti automatskog dovršavanja *kôda*, *refaktoriranja* i analize. Aplikacije razvijane korištenjem Android Studija nakon toga se *kompajliraju* u APK (*Android Package Kit*) format, spremne za distribuciju i objavljivanje na Google Play platformi. Koin je Kotlin biblioteka koja se koristi kako bi se ubrizgale ovisnosti, a rezultat toga je uspostavljanje fokusa na razvijanje aplikacije, a ne na alate. Također, Koin pruža jednostavne alate i API-je (*Application Programming Interface)* koji omogućuju izgradnju i integraciju Kotlin

tehnologija u aplikaciju omogućujući jednostavno skaliranje poslovanja [8].

Velika prednost Koina je što nudi jednostavan, lagan i prilagodljiv način upravljanja ovisnostima unutar aplikacije. Korisnost navedenog očituje se prilikom razvoja velikih aplikacija u kojima upravljanje ovisnostima postaje složenije [9].

Osim toga, Android Studio koristi značajku „*Apply Changes*“ koja omogućuje slanje izmjena *kôda* i resursa u aplikaciju koja se trenutno izvodi, što ubrzava iteracije tijekom razvoja.

Na službenim Googleovim stranicama moguće je preuzeti Android Studio i razvojni softverski paket (SDK) što programerima omogućuje jednostavan pristup svim potrebnim alatima za izradu visokokvalitetnih Android aplikacija [10].




***Slika 3.1.*** *Pregled sučelja Android Studija*

## 3.2. Kotlin

Kotlin je programski jezik otvorenog *kôda* koji podržava statičko tipiziranje, a to znači da se tipovi varijabli određuju u vrijeme prevođenja, a ne tijekom izvođenja te omogućuje primjenu kako objektno orijentiranih, tako i funkcionalnih programskih paradigmi. Njegova sintaksa i koncepti dijele puno sličnosti s jezicima poput C#, Jave i Scale. Iako Kotlin ne nastoji biti potpuno inovativan, inspiraciju vuče iz višegodišnjeg razvoja programskih jezika. Jezik je dostupan u verzijama prilagođenim za različite platforme, uključujući JVM (*Java Virtual Machine*) korištenjem Kotlin/JVM, JavaScript korištenjem Kotlin/JS i izvorni *kôd* korištenjem

Kotlin/Native.

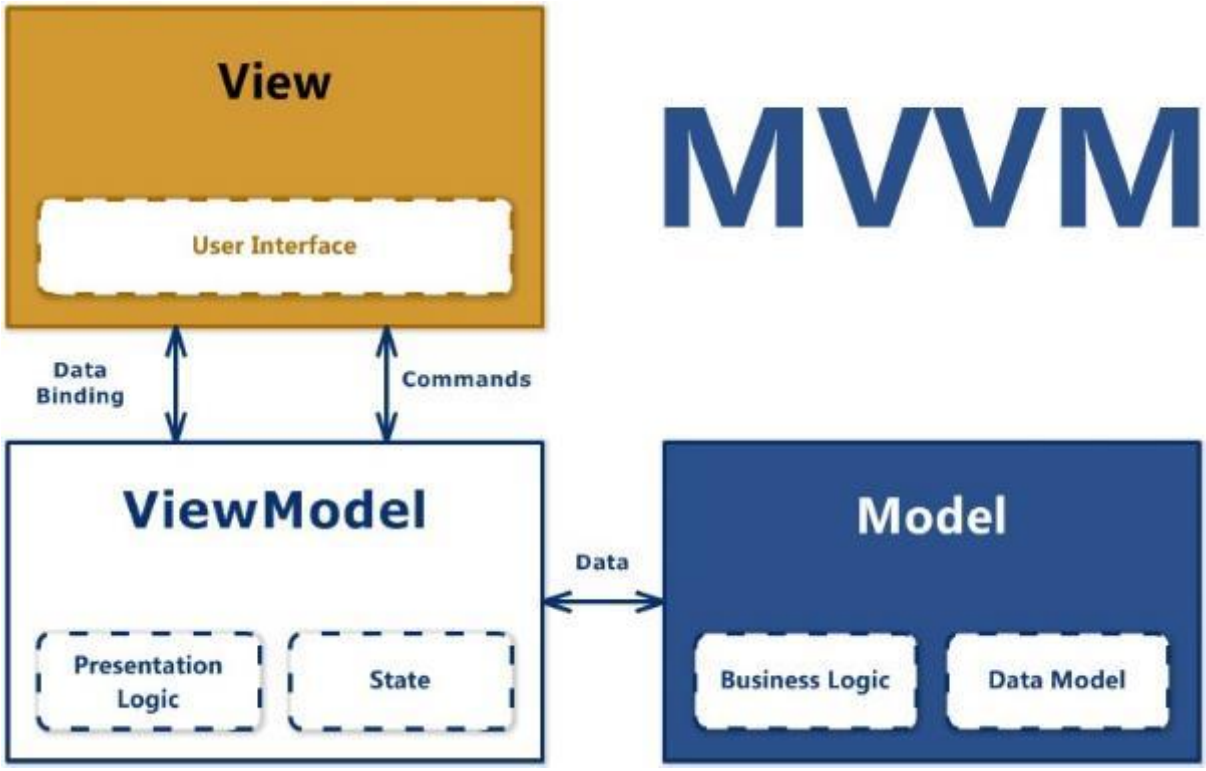
Kotlin Foundation, koju su osnovali JetBrains i Google, odgovorna je za razvoj i održavanje jezika s fokusom na njegovo kontinuirano unaprjeđivanje. Google je službeno uveo podršku za Kotlin u razvoju Android aplikacija, što znači da su dokumentacija i alati za Android prilagođeni radu s Kotlinom.

Interoperabilnost Kotlina s Javom ključna je za njegov rast i prihvaćanje među programerima. Omogućava pozivanje Java *kôda* iz Kotlina i obrnuto čime se može koristiti postojeća Java knjižnica. Popularnost Kotlina pridonosi ugodnijem razvojnim iskustvu na Androidu, dok se razvoj Android okvira nastavlja uz podršku oba jezika, Kotlin i Javu. Dok su neki Android API-ji, poput Android KTX-a (*Kotlin Extensions*), specifični za Kotlin, većina API-ja je napisana u Javi, ali se može koristiti i u Kotlinu pružajući fleksibilnost i široku podršku za različite programerske potrebe [11].

Kotlin podržava niz naprednih značajki uključujući funkcije višeg reda, anonimne funkcije, lambde, *inline* funkcije, *closures*, rekurziju s povratom i generičke tipove. Ukratko, Kotlin posjeduje sve funkcionalnosti i prednosti koje karakteriziraju funkcionalne programske jezike [12].

## 3.3. MVVM arhitekturni obrazac

Na slici 3.2. prikazan je arhitekturni obrazac MVVM punog naziva Model-View-ViewModel koji ima osnovnu podjelu na 3 komponente. Model upravlja podacima i poslovnom logikom, View se odnosi na grafičko korisničko sučelje, dok ViewModel djeluje kao posrednik između Modela i View-a. Ovaj pristup, preporučen od strane Googlea, omogućuje neovisni razvoj i testiranje svake komponente. Takva struktura povećava učinkovitost i smanjuje mogućnost pogrešaka osiguravajući stabilniji i održiviji razvoj aplikacija. Ova metoda ne samo da poboljšava organizaciju *kôda*, nego omogućava i lakše održavanje i proširivanje aplikacije tijekom njezinog životnog ciklusa [13-14].




***Slika 3.2.*** *Razdvajanje odgovornosti u MVVM arhitekturi [15]*

## 3.4. Firebase

Firebase je platforma koja djeluje kao BaaS (*Backend-as-a-Service*), odnosno poslužitelj kao usluga. Programerima se tako nudi raznovrstan skup alata i usluga za razvoj visokokvalitetnih aplikacija, povećanje baze korisnika i ostvarivanje profita. Temeljen na robusnoj Googleovoj infrastrukturi, Firebase se ističe kao pouzdano rješenje za moderne aplikacije.

Firebase se klasificira kao NoSQL (*Not Only Structured Query Language*) baza podataka koja pohranjuje podatke u dokumentima sličnim JSON formatu. U Firebaseu dokument predstavlja skup ključ-vrijednost parova definiranih shemom, dok zbirka dokumenata tvori kolekciju.

Neke od ključnih značajki Firebasea uključuju:

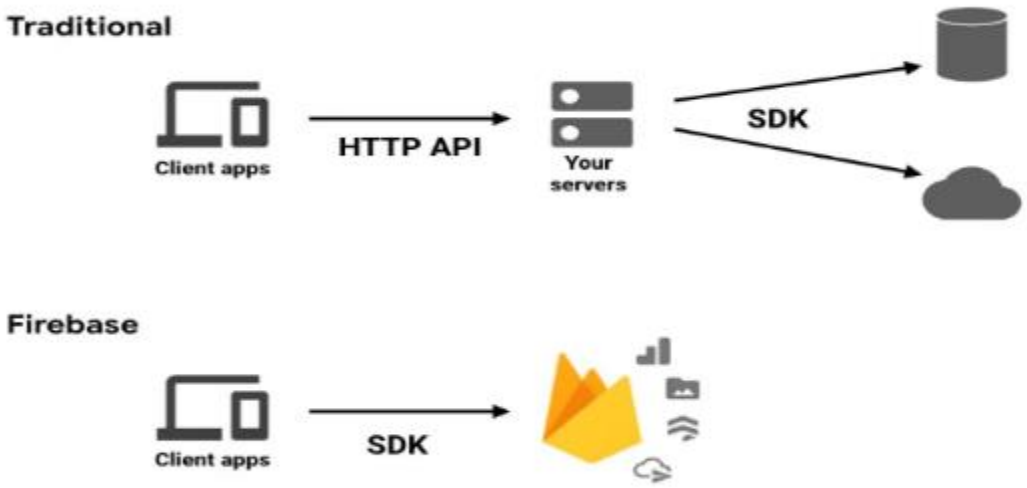
- **Autentifikacija**: Firebase omogućuje različite metode autentifikacije uključujući prijavu putem lozinki te telefonskih brojeva. Kao najveći bonus također je omogućena prijava putem usluga poput Googlea, Facebooka i Twittera što uvelike ubrzava cjelokupni proces. Pomoću Firebase Authentication SDK-a moguće je integrirati i prilagoditi više načina prijave u aplikaciju.
- **Baza podataka u stvarnom vremenu**: U stvarnom vremenu omogućena je

automatska sinkronizacija podataka među svim korisnicima koji ostaju dostupni i u offline načinu rada i na taj se način pruža neprestani pristup informacijama bez nepotrebnih prekida čak i u slučaju da aplikacija nema pristup internetu.

- **Hosting**: Firebase Hosting omogućuje brzo i učinkovito *hostanje* web aplikacija pri čemu se sadržaj predmemorira u globalnim mrežama za isporuku sadržaja.
- **Testni laboratorij**: Google nudi virtualne i fizičke uređaje smještene u njihovim podatkovnim centrima koji služe za testiranje aplikacija. Ova funkcionalnost omogućuje detaljnu provjeru kvalitete i osigurava kompatibilnost s raznim uređajima i platformama.
- **Obavijesti**: Koristeći Firebase moguće je slati obavijesti bez potrebe za dodatnim *kôdiranjem* što olakšava komunikaciju s korisnicima.

Ove značajke čine Firebase moćnim alatom za programere omogućujući im da se fokusiraju na stvaranje izvrsnih korisničkih iskustava, dok se Firebase brine za infrastrukturu i operativne zadatke. [16].

Slika 3.3. Prikazuje kako Firebase funkcionira u odnosu na tradicionalni načina razvoja aplikacije. Tradicionalni način obično uključuje pisanje softvera za *frontend* i *backend*. *Fronted kôd* samo poziva API „*endpoints*“ koje izlaže *backend,* dok će u stvarnosti *backend kôd* obaviti posao. Kada je riječ o Firebase proizvodima, oni uklanjaju potrebu za tradicionalnim *backendom* prebacujući odgovornost obrade podataka na klijentsku stranu. Administratorski pristup proizvodima za sve ove proizvode osiguran je putem *Firebase* konzole [17].




***Slika 3.3.*** *Razlika tradicionalnog načina razvoja aplikacije od Firebaseovog [17]*

## 3.5. Kotlin Korutine

Asinkrono ili neblokirajuće programiranje ključan je aspekt modernog razvoja softvera. Tijekom razvoja serverskih, desktop ili mobilnih aplikacija važno je osigurati da korisničko iskustvo bude fluidno i *responzivno*, a istovremeno omogućiti skalabilnost sustava prema potrebi.

Kotlin se ovom izazovu prilagođava na vrlo fleksibilan način pružajući podršku za korutine direktno na razini jezika, dok većinu funkcionalnosti delegira bibliotekama.

Korutine ne samo da olakšavaju asinkrono programiranje, već otvaraju i čitav niz drugih mogućnosti. Omogućuju istodobno izvršavanje više zadataka i implementaciju aktera, čime se dodatno unapređuje učinkovitost i performanse aplikacija [18].

## 3.6. Room baza podataka

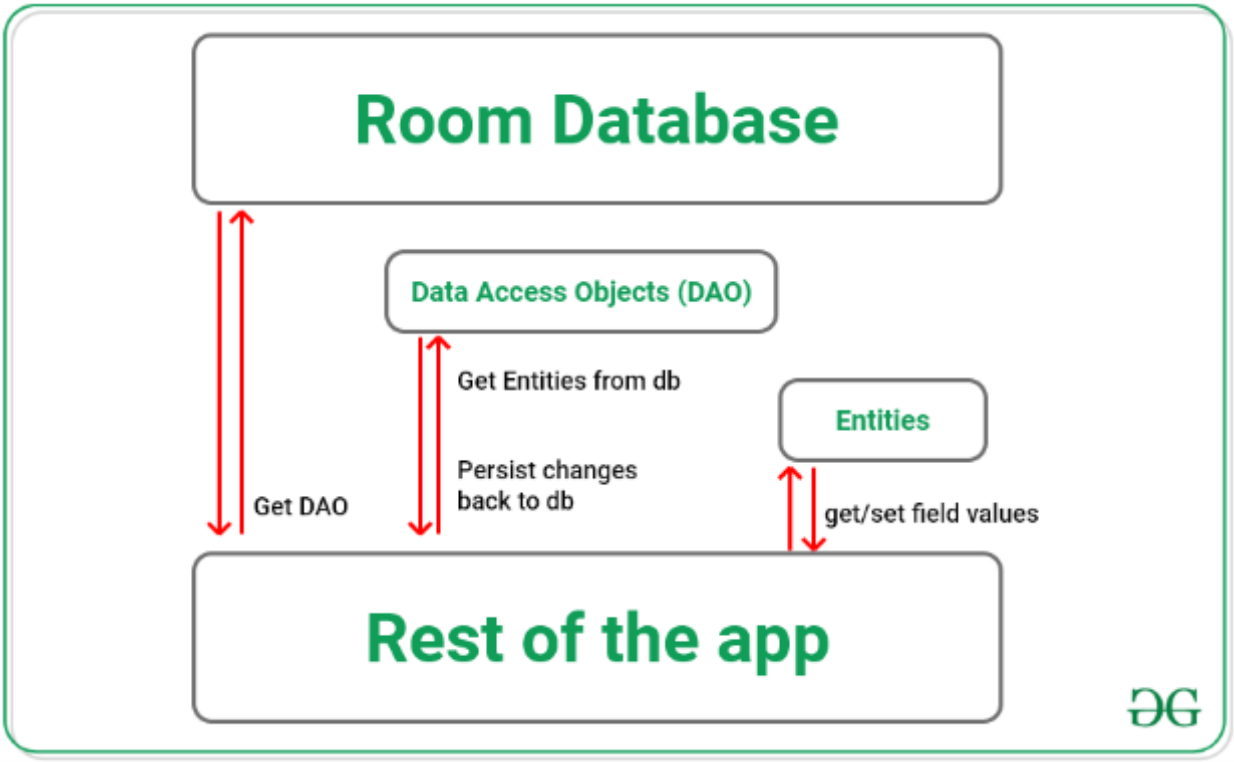
Aplikacije koje upravljaju značajnim količinama strukturiranih podataka mogu značajno profitirati od lokalnog pohranjivanja tih podataka. Najčešći primjer uporabe je *predmemoriranje* dijelova podataka koji su relevantni što omogućuje korisnicima pregledavanje sadržaja čak i kada uređaj nema pristup mreži.

Biblioteka Room za pohranu podataka pruža sloj apstrakcije iznad SQLite-a, omogućujući jednostavan pristup bazi podataka dok koristi sve mogućnosti SQLite-a. Na slici 3.4. prikazana je arhitektura Room baze koja ilustrira način na koji se komponente poput DAO-a (*Data Access Object*) i entiteta povezuju s ostatkom aplikacije. Room donosi niz pogodnosti, uključujući:

- Provjeru SQL upita tijekom vremena prevođenja.
- Praktične anotacije koje smanjuju ponavljajući i skloni greškama „*boilerplate*“ *kôd*.
- Učinkovite putanje za migraciju baze podataka.

Korištenjem Room biblioteke razvoj aplikacija koje zahtijevaju lokalnu pohranu podataka postaje puno jednostavniji i učinkovitiji pružajući stabilno i brzo iskustvo za krajnje korisnike [19].

Također, treba spomenuti da se korištenjem Room entiteta može definirati shema koja predstavlja bazu podataka programera, sve to bez pisanja SQL *kôda*. Tako se olakšava proces razvoja budući da se uklanja potreba za ručnim pisanjem i održavanjem SQL upita, čineći razvoj baze podataka bržim i manje sklonom greškama [20].

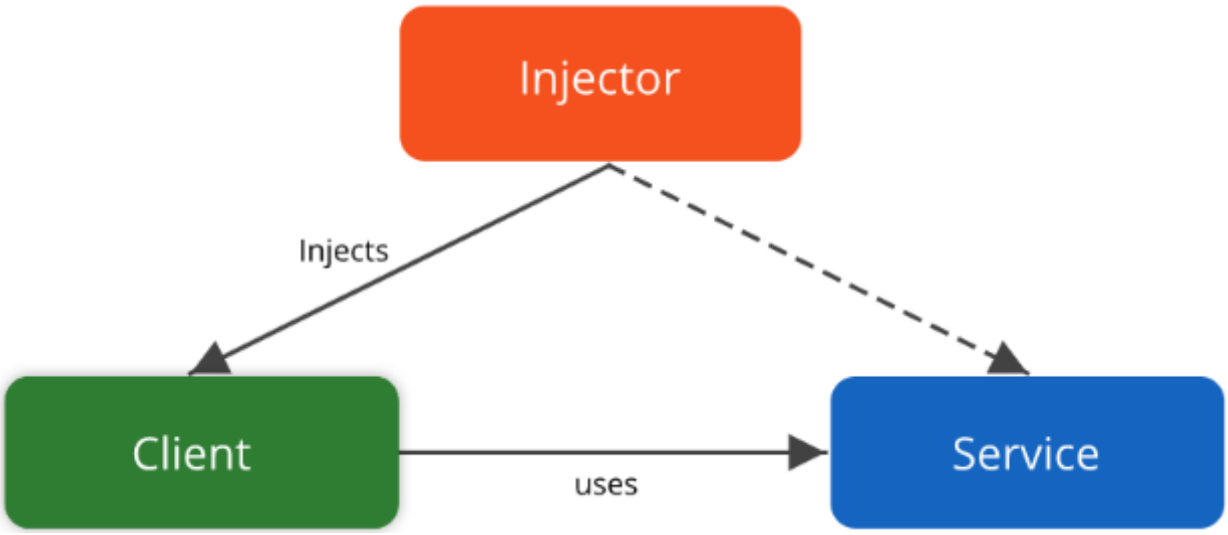



***Slika 3.4.*** *Arhitektura Room baze [21]*

## 3.7. Koin Dependency Injection

Kako bi se programeri mogli bolje usredotočiti na sam proces razvoja aplikacije, koristi se Koin – biblioteka specifična za ubrizgavanje ovisnosti u Kotlinu. Koin omogućuje učinkovito upravljanje ovisnostima smanjujući potrebu za složenim *kôdom* pri njihovom rukovanju. Na slici 3.5. prikazan je dijagram ubrizgavanja ovisnosti korištenjem Koin biblioteke, gdje „*Injector*“ upravlja povezivanjem klijenta s potrebnim uslugama. Uz to, Koin nudi jednostavne alate i API-je koji olakšavaju implementaciju i integraciju Kotlin tehnologija unutar aplikacije, omogućujući pritom lako skaliranje sustava [8].

Velika prednost Koina je što nudi jednostavan, lagan i prilagodljiv način upravljanja ovisnostima unutar aplikacije. Korisnost navedenog očituje se prilikom razvoja velikih aplikacija gdje upravljanje ovisnostima postaje složenije [9].

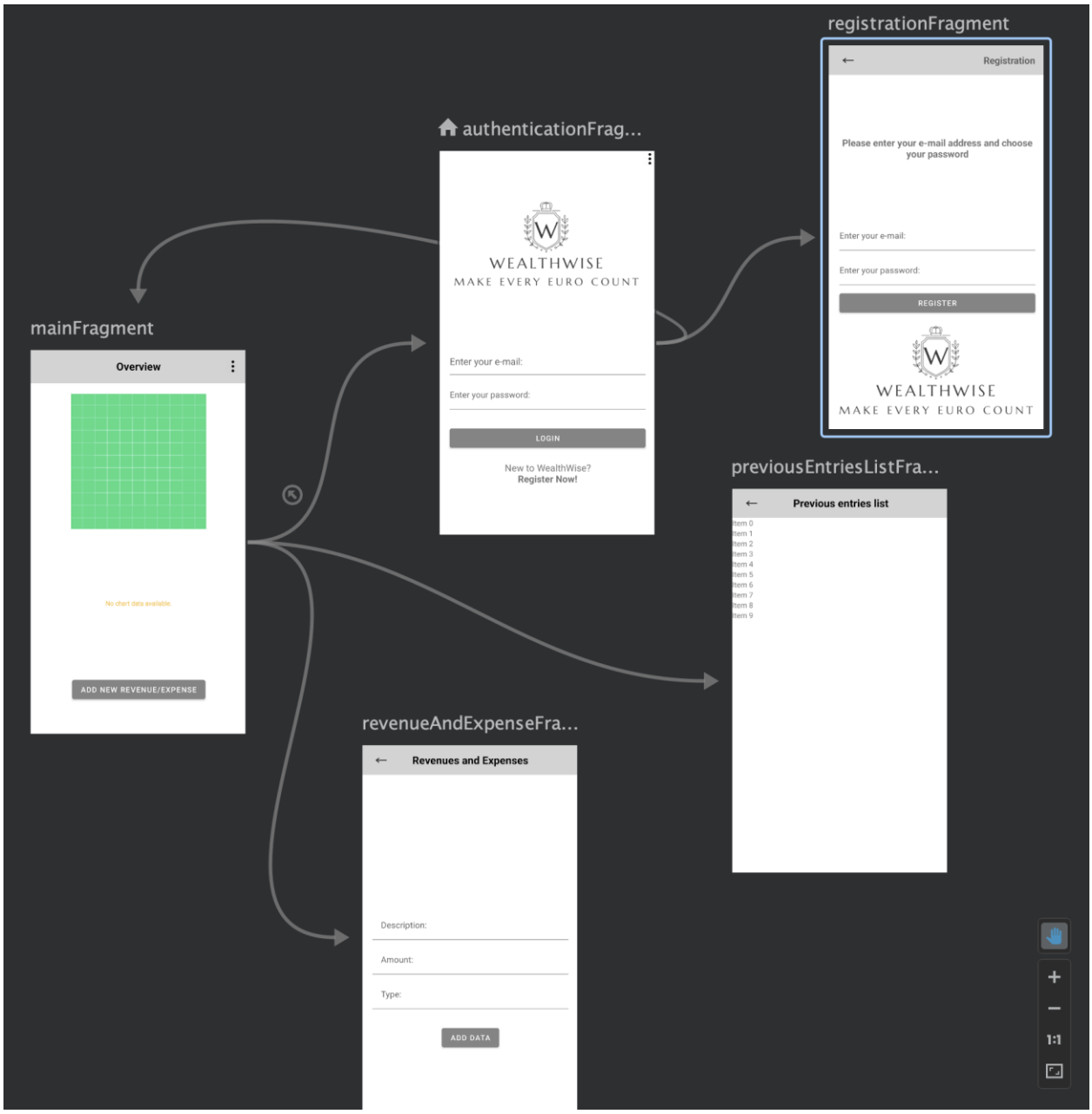



***Slika 3.5.*** *Dijagram ubrizgavanja Koin ovisnosti [22]*

## 3.8. Navigation Component

Navigacijska komponenta je biblioteka koja omogućuje učinkovito upravljanje složenom navigacijom unutar aplikacije. Ona podržava animacije prijelaza, duboko povezivanje i provjeru argumenata tijekom *kompajliranja* između različitih zaslona. Ova komponenta olakšava kreiranje fluidnog korisničkog iskustva i osigurava da se navigacija u aplikaciji odvija bez problema [23].

Na slici 3.6. prikazana je implementacija navigacijske komponente u „WealthWise“ aplikaciji koja ilustrira prijelaze između različitih fragmenata unutar aplikacije. Korištenjem navigacijske arhitekturne komponente korisnicima se pruža dosljedno i predvidljivo iskustvo bez potrebe za trošenjem vremena na održavanje *kôda*. Ova značajka pojednostavljuje razvoj aplikacije, uklanja nepotrebnu složenost i poboljšava korisničko iskustvo [24].




***Slika 3.6.*** *Grafički prikaz navigacije*

# 4. STRUKTURA I ARHITEKTURA SUSTAVA

U narednim dijelovima rada, bit će predstavljen detaljan uvid u strukturu i arhitekturu projekta. Bit će objašnjene najvažnije točke. Također, bit će prikazan dijagram toka aplikacije koji će vizualno predstaviti bitne korake i procese unutar aplikacije olakšavajući razumijevanje njenog djelovanja.

## 4.1. Struktura

Eksponencijalni rast složenosti i sve veći funkcionalni zahtjevi mobilnih aplikacija dovode do značajnih izazova u pogledu navigacije i upravljanja projektima. Android Studio nudi napredne mogućnosti za sistematizaciju projekata putem kreiranja modularnih paketa koji omogućuju efektivno organiziranje i raspodjelu međusobno povezanih klasa i datoteka. Ovi paketi, prikazani kao direktoriji, pružaju jednostavnije održavanje i unaprijeđenu preglednost projekta. Postizanje optimalne organizacije i operativne efikasnosti ostvaruje se kroz pažljivo osmišljen i jasno definiran raspored paketa unutar Android projekta.

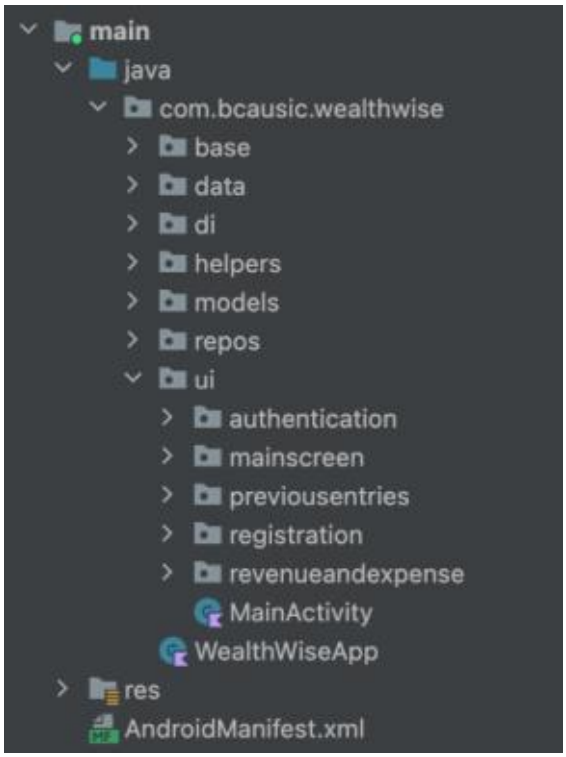
Slika 4.1. prikazuje primjer strukture projekta „WealthWise“ koji ilustrira hijerarhijsku organizaciju u kojoj su su različiti aspekti funkcionalnosti smješteni u specifične pakete. Takva struktura omogućuje programerima efikasnije upravljanje *kôdom*, lakše pronalaženje specifičnih klasa i poboljšanu modularnost prilikom implementacije novih značajki ili održavanja postojećih. Ova praksa ne samo da unapređuje preglednost, već i doprinosi smanjenju potencijalnih pogrešaka i povećanju efikasnosti razvojnih procesa. U konačnici, dobro organiziran projekt uvelike olakšava rad razvojnim timovima omogućujući im fokus na inovacije i kvalitetu.

Prema slici 4.1., postoji nekoliko ključnih paketa u projektu:

- **base**: Ovaj paket sadržava temeljne klase i funkcije koje čine osnovu funkcionalnosti za cijeli projekt osiguravajući stabilnost i konzistentnost u radu aplikacije.
- **data**: Unutar ovog paketa nalaze se sve podatkovne klase koje su ključne za upravljanje podacima unutar aplikacije.
- **di**: Paket namijenjen ubrizgavanju ovisnosti (*Dependency Injection*) uključuje module koji omogućuju efikasno povezivanje različitih dijelova aplikacije smanjujući međusobnu ovisnost između klasa i povećavajući modularnost.
- **helpers**: U ovom paketu smještene su pomoćne datoteke uključujući konstante i globalne pružatelje dispečera za korutine koje značajno olakšavaju i optimiziraju

različite operacije unutar aplikacije.

- **models**: Paket koji sadržava sve modele poslovne logike potrebne za ispravno funkcioniranje aplikacije.
- **repos**: Repozitoriji za unos i pohranu podataka koji omogućuju efikasno upravljanje podacima osiguravajući njihovu dostupnost i konzistentnost.
- **ui**: Predstavlja paket koji sadrži sve datoteke i klase koje su povezane s korisničkim sučeljem. Unutar njega nalaze se „*potpaketi*“ koji specificiraju različite zaslone aplikacije:
    - **authentication**: Paket posvećen klasama vezanim za autentifikaciju korisnika, pružajući sigurnosne mehanizme za prijavu i registraciju.
    - **mainscreen**: Paket koji uključuje klase za glavni zaslon aplikacije omogućujući korisniku pristup glavnim funkcionalnostima.
    - **previousentries**: Sadrži klase koje omogućuju pregled prethodnih unosa korisnika osiguravajući povijest transakcija i aktivnosti.
    - **registration**: Paket sadrži sve datoteke vezane za proces registracije novih korisnika.
    - **revenueandexpense**: Paket namijenjen upravljanju prihodima i rashodima uključujući klase za unos i pregled financijskih podataka čime se korisnicima omogućuje praćenje njihovih financijskih aktivnosti.




***Slika 4.1.*** *Projektna struktura paketa*

## 4.2. Dijagram Toka

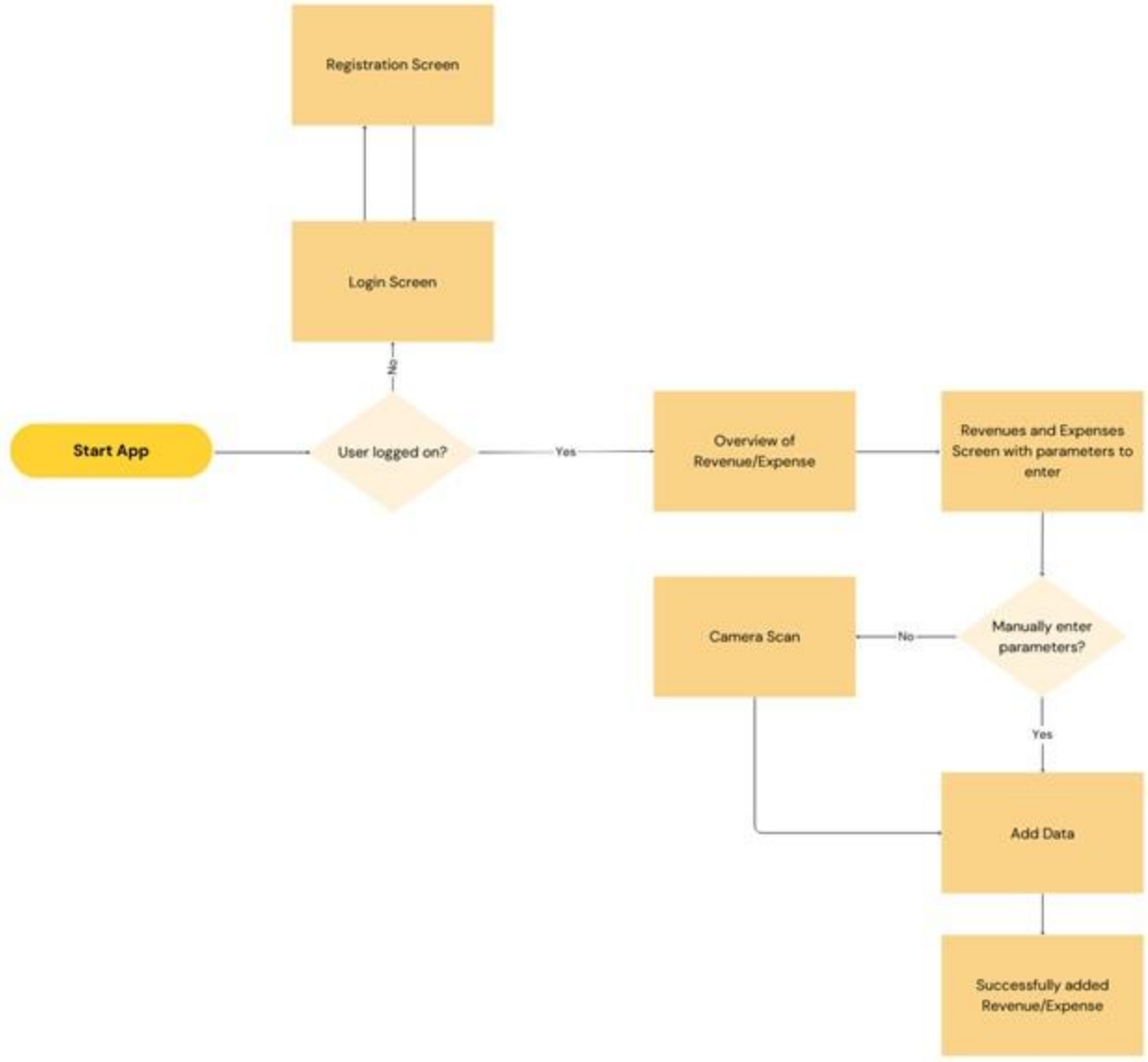
Dijagram toka predstavlja vizualizaciju odvojenih koraka unutar procesa prikazanih u sekvencijalnom redoslijedu. Kao univerzalni alat može se prilagoditi različitim namjenama uključujući proizvodne, administrativne i uslužne procese kao i planove projekata. Često se koristi za analizu procesa i prepoznat je kao jedan od sedam osnovnih alata kvalitete.

U dijagram toka mogu biti uključeni različiti elementi kao što su redoslijed radnji, materijali ili usluge koje ulaze u proces ili izlaze iz njega (ulazi i izlazi), odluke koje treba donijeti, sudionici u procesu, vrijeme potrebno za svaki korak i mjerenja procesa.

Dijagram toka se koristi za razvijanje razumijevanja kako se proces odvija, za proučavanje procesa radi poboljšanja, za komunikaciju s drugima o tijeku procesa, za dokumentiranje procesa i za planiranje projekta. Umjesto da se zadatak crtanja dijagrama povjeri tehničkom stručnjaku, najbolje je da osobe koje zapravo obavljaju proces sudjeluju u njegovoj izradi. Time se osigurava točnost i praktičnost dijagrama.

Prilikom izrade dijagrama toka nije nužno slijediti stroga pravila crtanja. Najvažnije je da dijagram pomaže sudionicima u razumijevanju procesa i njegovom poboljšanju [25].

Na slici 4.2. prikazan je dijagram toka za dodavanje novih prihoda i rashoda u aplikaciji „WealthWise“**,** gdje se jasno može vidjeti korak po korak procesa počevši od pokretanja aplikacije, preko prijave korisnika pa sve do uspješnog unosa financijskih podataka. Dijagram obuhvaća i odluke poput odabira između ručnog unosa podataka ili korištenja skeniranja putem kamere, čime se pruža fleksibilnost korisnicima prilikom dodavanja novih stavki u aplikaciju.




***Slika 4.2**. Dijagram toka za dodavanje novih prihoda/rashoda*

## 4.3. Arhitektura

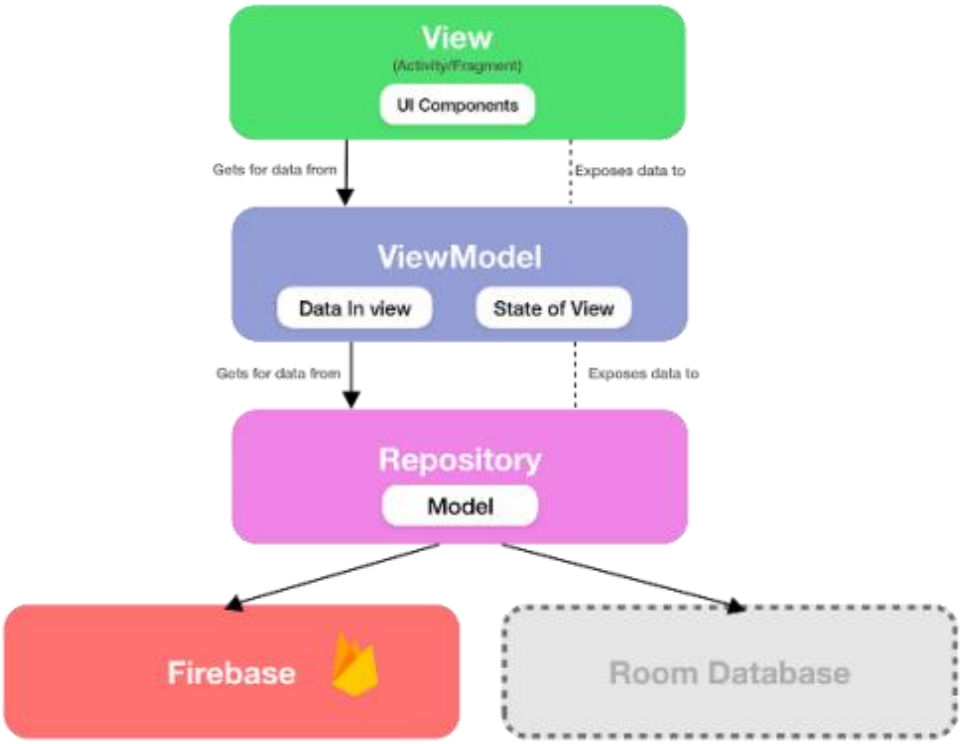
Za implementaciju ove aplikacije odabrana je moderna i vrlo raširena MVVM arhitektura. Korištenjem MVVM arhitekture i razdvajanjem *kôda* u različite slojeve koji su zaduženi za specifične dijelove toka podataka, moguće je stvoriti čist, održiv i lako upravljiv *kôd*.

Kao što je vidljivo i na slici 4.3., u tradicionalnom MVVM obrascu različiti slojevi toka podataka su:

- **Model**: Koristi se za modeliranje podataka.
- **View**: Koristi se za prikaz podataka korisniku.
- **View-Model**: Koristi se za modeliranje podataka/stanja koje bi prikaz trebao

prikazati.

Na taj način MVVM omogućuje jasan i strukturiran razvoj softverskih rješenja čineći *kôd* preglednim i lakim za održavanje [26].



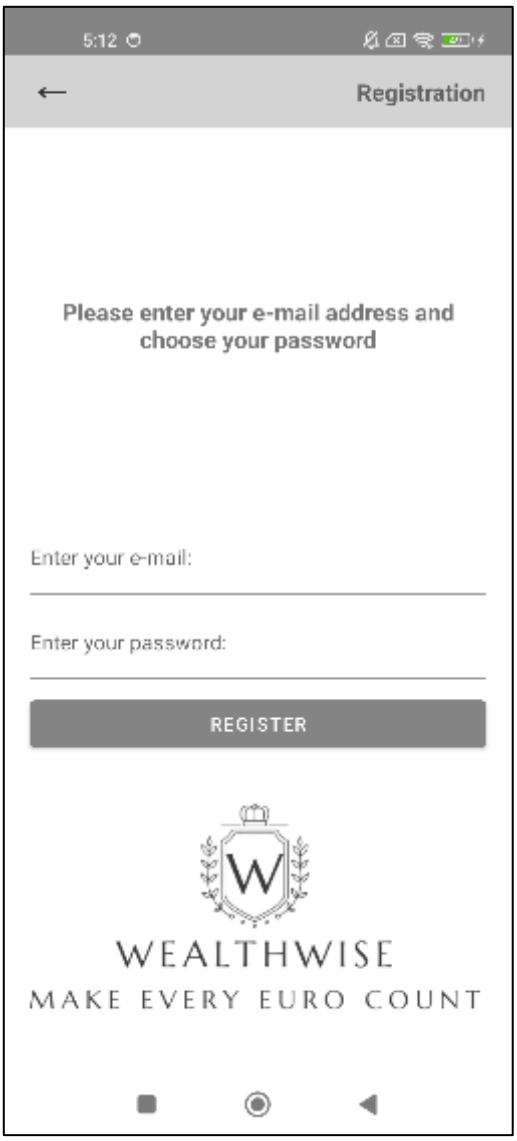

***Slika 4.3.*** *Arhitektura (MVVM) projekta [26]*

# 5. PROGRAMSKO RJEŠENJE I IZGLED APLIKACIJE

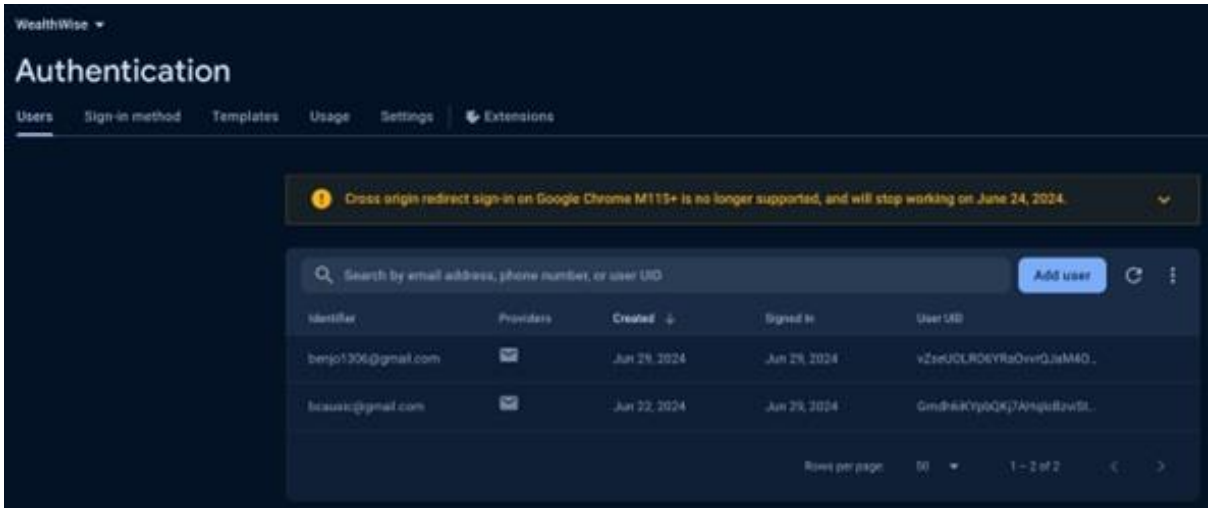
U narednom poglavlju detaljno će se razjasniti metode i postupci korišteni pri izradi aplikacije kao i način na koji svi dijelovi sustava međusobno funkcioniraju. Ovaj dio rada pružit će dublji uvid u tehnologije korištene za razvoj specifičnih funkcionalnosti koje omogućuju nesmetani rad aplikacije. Detaljna analiza omogućit će razumijevanje tehničkih aspekata te kako se oni uklapaju u cjelokupni sustav osiguravajući učinkovitost i pouzdanost aplikacije.

## 5.1. Registracija i prijava korisnika

Dizajniranje i implementacija obrasca za autentifikaciju bilo je neophodno kako bi se omogućila registracija i prijava korisnika. Koristeći napredne mogućnosti Firebase Authentication usluge korisnicima je omogućeno da se prijavljuju i registriraju putem e-mail adrese i lozinke. Slika 5.1. prikazuje fragment za registraciju. Nakon što su se upisali svi potrebni podaci, uključujući validan email i validnu lozinku, korisnik će se uspješno registrirati klikom na gumb „*REGISTER*“. Slika 5.2. prikazuje kako Firebase Firestore pohranjuje dokumente u kolekciju „*users*“, dodjeljujući jedinstveni identifikator svakom korisniku. Ovaj sustav autentifikacije osigurava sigurnost i jednostavnost korištenja omogućavajući korisnicima glatku interakciju s aplikacijom, dok istovremeno osigurava čuvanje njihovih podataka u oblaku putem Firebase servisa. Korištenje ovih tehnologija značajno unapređuje korisničko iskustvo te povećava efikasnost i pouzdanost cijelog procesa prijave i registracije.




***Slika 5.1.*** *Izgled fragmenta za registraciju*




***Slika 5.2.*** *Izgled Firebase Authentication-a*

Slika 5.3. metoda setOnClickListeners() inicijalizira događaje za povratak na prethodni zaslon (sivBack) i registraciju (btnRegister). Klikom na btnRegister aplikacija pokreće metodu createNewAccount() u RegistrationViewModel klasi koja započinje proces registracije korisnika.

```
private fun setOnClickListeners() {
    binding.sivBack.setOnClickListener { it: View!
        navigateToAuthentication()
    }

    binding.btnRegister.setOnClickListener { it: View!
        shouldShowProgressDialog(shouldShowProgress = true)
        registrationViewModel.createNewAccount(
            binding.textInputEditTextEmail.text.toString(),
            binding.textInputEditTextPassword.text.toString()
        )
    }
}
```

***Slika 5.3.*** *Prikaz postavljanja događaja za navigaciju i registraciju*

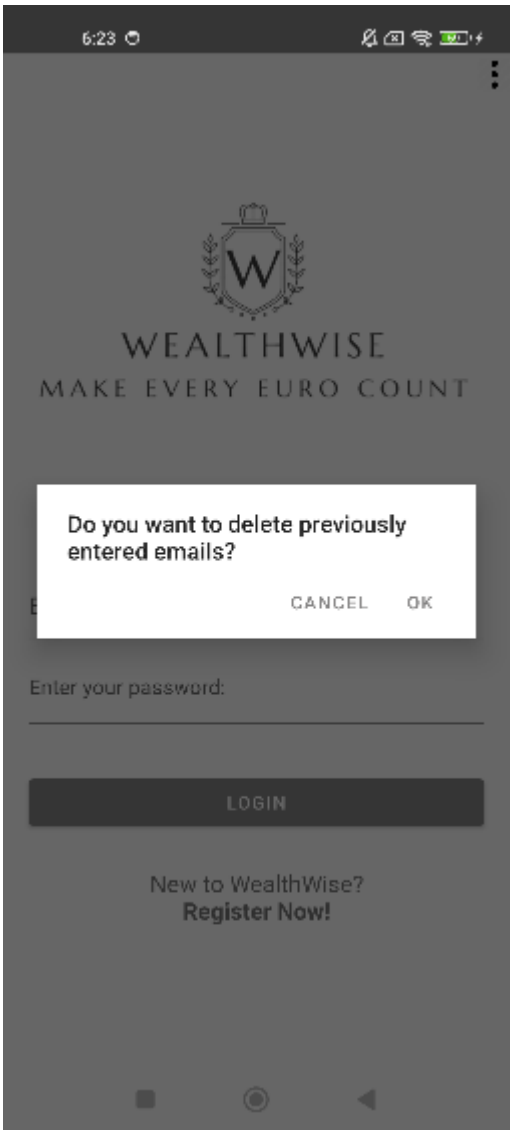
Slika 5.4. prikazuje metodu createNewUserWithEmailAndPassword() koja koristi Firebase Authentication API za stvaranje novog korisničkog računa. Metoda koristi *suspend* funkciju, a to znači da je ova funkcija suspendirana, odnosno može biti pozvana iz korutina ili drugih suspendiranih funkcija. Ovaj pristup omogućava asinkrono programiranje bez blokiranja glavne niti budući da će se metode izvršavati u pozadinskoj niti. Uspješan rezultat prikazuje se korisniku putem tost poruke, dok se neuspjeh evidentira i također obavještava korisnika.


***Slika 5.4.*** *Prikaz kreiranja novog korisničkog računa putem emaila*

Slika 5.5. prikazuje početni zaslon aplikacije. Na vrhu zaslona nalazi se logotip aplikacije, dok se u gornjem desnom kutu nalaze tri točkice koje predstavljaju izbornik za dodatne opcije. Kada korisnik klikne na tri točkice prikazuje se „*alert dialog*“ sa pripadajućom porukom kao što je prikazano na slici 5.6. Ova poruka omogućuje korisniku da obriše prethodno unesene e-mail adrese koje su zapamćene tijekom ranijih prijava. Ako korisnik odabere opciju „*OK*“ svi prethodno zapamćeni e-mailovi bit će obrisani, a automatska ispunjavajuća funkcija više neće nuditi te e-mail adrese prilikom upisivanja.

Nakon što je korisnik uspješno registriran, potrebno je unijeti e-mail adresu i lozinku u odgovarajuća polja za prijavu, „*Enter your e-mail*:“ i „*Enter your password*:“ kako bi se pristupilo daljnjim funkcionalnostima aplikacije. Ovaj proces osigurava da korisnik može sigurno i jednostavno pristupiti aplikaciji koristeći svoje autentifikacijske podatke.


***Slika 5.5.*** *Izgled početnog zaslona „AuthenticationFragment”*




***Slika 5.6.*** *„initAlertDialog()“ dijalog s porukom upozorenja*

Slika 5.7. prikazuje metodu signIn(). Ova funkcija koristi podatke koje je korisnik unio (e-mail, lozinka) za autentifikaciju preko Firebase Authentication servisa.

```
fun signIn(email: String, password: String) {
    viewModelScope.launch { this: CoroutineScope
        firebaseAuthRepository.authenticateUserWithEmail(email, password) { it: Boolean
            _isUserSignedIn.postValue(it)
        }
    }
}
```

***Slika 5.7.*** *Prikaz autentifikacije korisnika putem emaila i lozinke*

Slika 5.8. prikazuje metodu isUserAlreadySignedIn(). Metoda koristi „viewModelScope.launch“ kako bi se autentifikacija izvršila u pozadinskoj niti čime se

izbjegava blokiranje glavne niti korisničkog sučelja. Rezultat autentifikacije ažurira „_isUserSignedIn“ vrijednost koja se promatra u AuthenticationFragment-u. Ako prijava ne uspije, korisnik će dobiti poruku o pogrešci koju vraća Firebase. Ova poruka se prikazuje korisniku na zaslonu kako bi bio obaviješten o razlogu neuspjeha.

```
fun isUserAlreadySignedIn() {
    viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
        val currentFirebaseUser = firebaseAuthRepository.getCurrentUser()

        if(currentFirebaseUser != null) {
            _isUserSignedIn.postValue( value: true)
            SharedPrefsManager().saveUserId(currentFirebaseUser.uid)
        } else {
            _isUserSignedIn.postValue( value: false)
        }
    }
}
```

***Slika 5.8.*** *Prikaz provjere je li korisnik već prijavljen*

Slika 5.9. prikazuje metodu navigateToMainScreen(). Takav oblik metode se nalazi u svim klasama kako bi se omogućila intuitivna i efikasna tranziciju između zaslona, što doprinosi boljim korisničkim interakcijama unutar aplikacije. Kada je korisnik uspješno prijavljen ova metoda koristi funkciju „NavController“ za navigaciju do glavnog zaslona aplikacije. Ova navigacija omogućuje korisniku pristup ključnim funkcionalnostima aplikacije za upravljanje prihodima i rashodima osiguravajući neometano korisničko iskustvo i kontinuiranu upotrebu aplikacije.

```
private fun navigateToMainScreen() {
    findNavController().navigate(
        AuthenticationFragmentDirections.actionAuthenticationFragmentToRevenuesAndExpensesFragment()
    )
}
```

***Slika 5.9.*** *Prikaz navigacije na glavni zaslon aplikacije*

Slika 5.10. funkcija authenticateUserWithEmail() igra ključnu ulogu u autentifikaciji korisnika prilikom pokušaja prijave putem njihove e-mail adrese i lozinke koristeći Firebase Authentication servis.

U slučaju uspješne prijave „addOnSuccessListener“ postavlja „onResult(*true*)“ signalizirajući uspješnu autentifikaciju te sprema korisnički ID u „SharedPrefsManager“. Također, korisnici su preusmjereni na glavni zaslon aplikacije.

Ako autentifikacija ne uspije „addOnFailureListener“ postavlja „onResult(*false*)“ i

prikazuje poruku o pogrešci koristeći „makeToast“ funkciju.

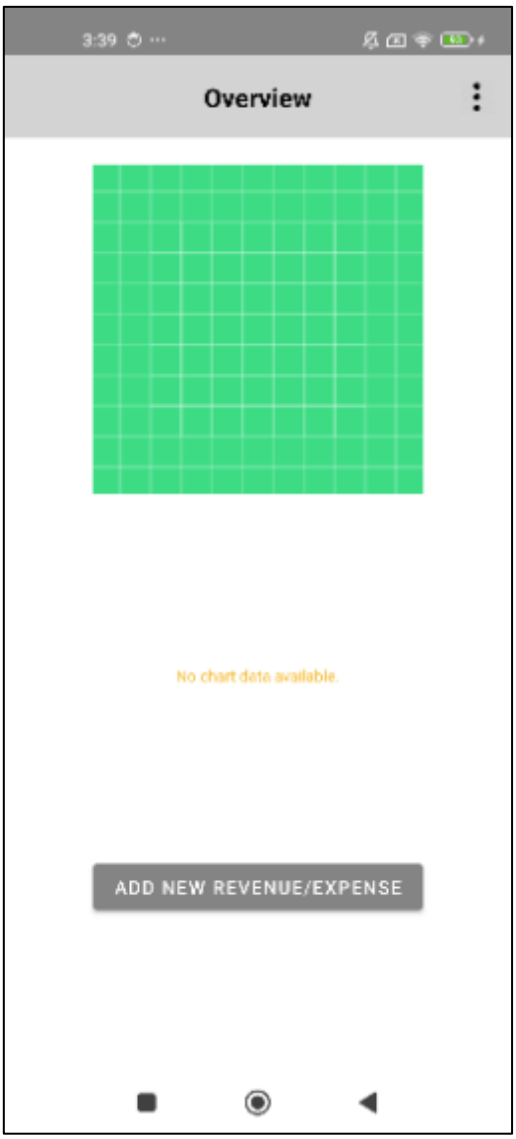
```
suspend fun authenticateUserWithEmail(email: String, password: String, onResult: (Boolean) -> Unit) {
    withContext(Dispatchers.IO) { this: CoroutineScope
        try {
            firebaseAuth.signInWithEmailAndPassword(email, password)
                .addOnSuccessListener { it: AuthResult!
                    onResult(true)
                    it.user?.let { firebaseUser ->
                        SharedPrefsManager().saveUserId(firebaseUser.uid)
                    }
                }
                .addOnFailureListener { it: Exception
                    it.printStackTrace()
                    onResult(false)
                    makeToast(it.message.toString(), lengthLong = false)
                }.await() ^withContext
        } catch (e: Exception) {
            onResult(false) ^withContext
        }
    }
}
```

***Slika 5.10.*** *Prikaz autentifikacije korisnika putem Firebasea*

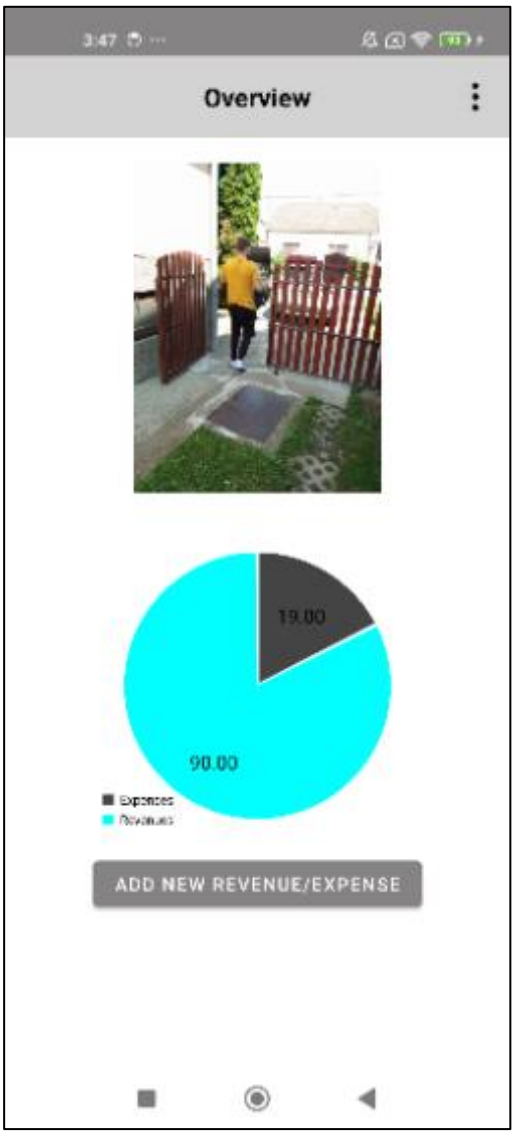
## 5.2. Glavni zaslon

Slika 5.11. prikazuje izgled glavnog zaslona prilikom korisnikovog prvog pristupa aplikaciji. Navedeni zaslon omogućuje korisnicima dodavanje novih prihoda i rashoda, prikazuje grafički prikaz (kružni grafikon) unesenih transakcija te prikaz profilne slike korisnika. U desnom gornjem kutu nalaze se tri točkice koje predstavljaju dodatne opcije čija će funkcionalnost biti detaljnije objašnjena u kasnijem dijelu rada. Na vrhu zaslona nalazi se naslov „*Overview*“. Ispod naslova nalazi se prostor predviđen za prikaz slike koji je trenutno prazan i označen zelenim okvirom. Klikom na zeleni okvir korisniku se nudi izbor između „*Camera*“ ili „*Gallery*“ za dodavanje slike. Ispod okvira nalazi se poruka „No chart data available.“ kojom sugerira da trenutačno nema dostupnih podataka za prikaz u grafičkom obliku. Na dnu zaslona smješten je gumb „*ADD NEW REVENUE/EXPENS*E“ koji korisniku omogućuje dodavanje novih prihoda ili rashoda. Nakon dodavanja prihoda ili rashoda pojavit će se kružni grafikon (*pie chart*).

Slika 5.12. prikazuje isti zaslon s već unesenim podacima. Na vrhu zaslona, ispod naslova „Overview“, nalazi se slika korisnika. Ispod slike prikazan je kružni grafikon koji vizualizira udjele prihoda i rashoda u trenutnom mjesecu. Plavi dio grafikona predstavlja prihode (90.00), dok sivi dio predstavlja rashode (19.00). Ova vizualizacija pomaže korisniku da brže i lakše razumije strukturu svojih financija. Na dnu zaslona nalazi se isti gumb „*ADD NEW REVENUE/EXPENSE*“.




***Slika 5.11.*** *Izgled glavnog zaslona „MainFragment“ bez podataka*




***Slika 5.12.*** *Izgled glavnog zaslona „MainFragment“ nakon unešenih podataka*

Slika 5.13. prikazuje metodu startCamera() koja korisnicima omogućuje pokretanje kamere uređaja za snimanje fotografija. U okviru ove metode kreira se „Intent“ s akcijom „MediaStore.ACTION_IMAGE_CAPTURE“ koja pokreće aplikaciju kamere na uređaju. „Intent**“** je mehanizam za komunikaciju između različitih komponenti aplikacije ili aplikacija na Androidu. Nakon što se slika snimi, rezultat se vraća u onActivityResult() metodu pomoću identifikatora „CAMERA_CODE“.

```
private fun startCamera() {
    val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    startActivityForResult(cameraIntent, CAMERA_CODE)
}
```

***Slika 5.13.*** *Prikaz pokretanja kamere za snimanje fotografija*

Slika 5.14. prikazuje metodu startGallery() koja korisnicima omogućuje odabir slike iz galerije uređaja. Kreira „Intent“ s tipom sadržaja „image/*“ i akcijom „Intent.ACTION_GET_CONTENT“ koja otvara izbor aplikacija za odabir slike. Nakon što korisnik odabere sliku, rezultat se vraća u onActivityResult() metodu pomoću identifikatora „GALLERY_CODE“.

```
private fun startGallery() {
    val pickIntent = Intent()
    pickIntent.type = "image/*"
    pickIntent.action = Intent.ACTION_GET_CONTENT
    startActivityForResult(Intent.createChooser(pickIntent,  title: "Select Picture"), GALLERY_CODE)
}
```

***Slika 5.14.*** *Prikaz odabira slike iz galerije uređaja*

Slika 5.15. prikazuje metodu onActivityResult() koja se poziva nakon završetka aktivnosti i povratka rezultata u fragment. Ovdje se provjerava je li rezultat došao iz galerije ili kamere te je li operacija bila uspješna koristeći „Activity.RESULT_OK“. Ako je rezultat došao iz galerije, dobivena slika se učitava pomoću URI-ja slike i zatim se prosljeđuje u ViewModel (putem „mainFragmentViewModel“ instance) za daljnju obradu kao što je prijenos slike na cloud storage. Ako je rezultat došao iz kamere, dobivena slika se također prosljeđuje u ViewModel nakon što se pretvori u URI, omogućujući dodatne operacije poput spremanja slike ili njezine daljnje manipulacije unutar aplikacije. Ovaj postupak omogućuje korisniku jednostavno učitavanje i korištenje slika unutar aplikacije bez potrebe za ručnim upravljanjem datotekama.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(requestCode == GALLERY_CODE && resultCode == Activity.RESULT_OK) {
        if(data != null) {
            val takenImage = data.data
            if (takenImage != null) {
                mainFragmentViewModel.uploadAvatar(takenImage)
                binding.sivAvatar.load(takenImage)
            }
        }
    }
    else if(requestCode == CAMERA_CODE && resultCode == Activity.RESULT_OK) {
        if (data != null && data.hasExtra( name: "data")) {
            val takenImage = mainFragmentViewModel.getImageUri(requireContext(), data.extras?.get("data") as Bitmap)
            mainFragmentViewModel.uploadAvatar(takenImage)
            binding.sivAvatar.load(takenImage)
        }
    }
}
```

***Slika 5.15.*** *Prikaz obrade rezultata odabira slike ili fotografije*

Slika 5.16. prikazuje metodu uploadAvatar() koja inicira prijenos fotografije na

Firebase Cloud Storage. Parametar „photoUri“ predstavlja URI slike koja se prenosi. Metoda koristi „viewModelScope.launch“ kako bi osigurala da se operacija prijenosa izvršava u pozadinskoj korutini čime se izbjegava blokiranje glavne niti.

```
fun uploadAvatar(photoUri: Uri) {
    viewModelScope.launch { this: CoroutineScope
        firebaseCloudStorageRepository.uploadPhoto(photoUri)
    }
}
```

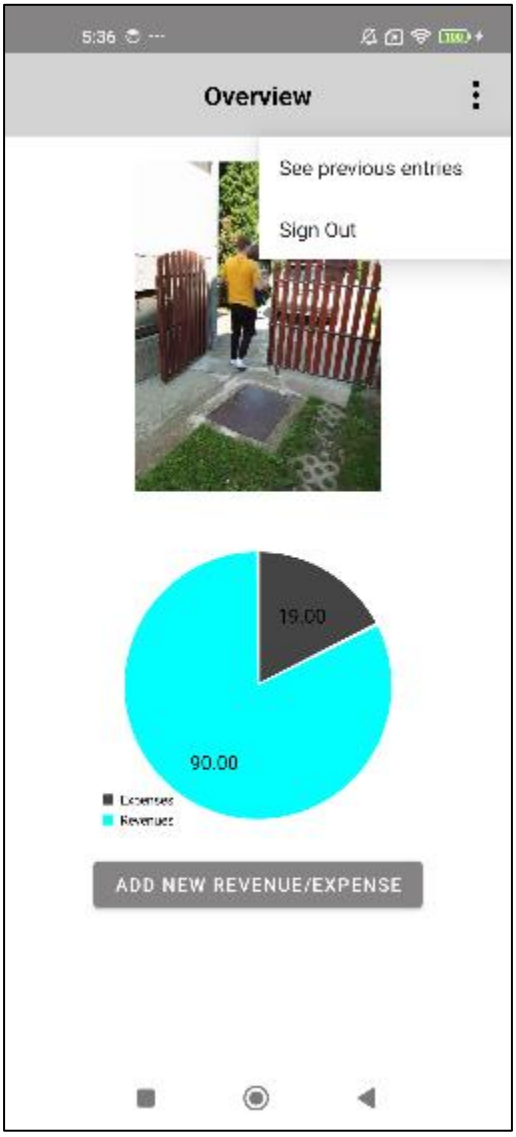
***Slika 5.16.*** *Prikaz prijenosa fotografije na Firebase Cloud Storage*

Slika 5.17. prikazuje metodu downloadAvatar() koja preuzima korisnički avatar iz Firebase Cloud Storagea. Ova metoda koristi korutine za asinkrono izvršavanje zadatka preuzimanja pozivajući funkciju downloadPhoto iz repozitorija FirebaseCloudStorageRepository koja dohvaća URI avatara i postavlja ga u „_userAvatar“ varijablu.

```
fun downloadAvatar() {
    viewModelScope.launch { this: CoroutineScope
        firebaseCloudStorageRepository.downloadPhoto { it: Uri
            _userAvatar.postValue(it)
        }
    }
}
```

***Slika 5.17.*** *Prikaz preuzimanja korisničkog avatara iz Firebasea*

Na slici 5.18. prikazano je kako se, nakon klika na tri točkice u gornjem desnom kutu zaslona, otvara padajući izbornik s opcijama „*See previous entries*“ i „*Sign Out*“. Ovaj izbornik omogućuje korisnicima pristup dodatnim funkcijama aplikacije. Klikom na prvu opciju korisnik se preusmjerava na zaslon s prethodno dodanim unosima koristeći već objašnjeni „NavController“. Klikom na drugu opciju korisnik se odjavljuje iz aplikacije.




***Slika 5.18.*** *Padajući izbornik*

Slika 5.19. prikazuje metodu koja poziva funkciju signOut() koja odjavljuje korisnika iz Firebase Authentication servisa, uklanja korisnički ID iz „SharedPrefsManager“ i postavlja vrijednost „_isUserSignedOut“ na „*true*“, što aplikaciji signalizira da je korisnik odjavljen.

```
fun signOut() {
    firebaseAuthRepository.signOut()
    SharedPrefsManager().saveUserId( id: "")
    _isUserSignedOut.postValue( value: true)
}
```

***Slika 5.19.*** *Prikaz odjave korisnika iz aplikacije*

## 5.3. Dodavanje prihoda/rashoda

U aplikaciji je omogućeno dodavanje novih prihoda i rashoda klikom na gumb „*ADD NEW REVENUE/EXPENSE*“ koji se nalazi na glavnom zaslonu. Nakon odabira ovog gumba korisniku se prikazuje novi zaslon prikazan na slici 5.20. koji omogućuje manualni unos opisa, iznosa i tipa transakcije. Polja „*Description*“, „*Amount*“ i „*Type*“ omogućuju korisnicima unos potrebnih podataka o financijskoj transakciji. Naravno, moguća su dva tipa podataka. Klikom na „*Type*“ otvorit će se padajući izbornik u kojem se odabere „*Revenue*“ ili „*Expense*“. Kada su svi podaci uneseni, pritiskom na gumb „*ADD DATA*“, unos se automatski sprema te se ažurira kružni grafikon kako bi prikazao nove financijske informacije.


***Slika 5.20.*** *Manualni unos podataka*

Osim ručnog unosa, aplikacija nudi i mogućnost skeniranja QR *kôda* pritiskom na gumb „*SCAN*“. U tom trenutku otvara se kamera uređaja s aktiviranim QR skenerom

omogućujući korisniku brzo i jednostavno automatsko popunjavanje svih potrebnih polja temeljenih na skeniranom QR *kôdu*. Na slici 5.21. prikazan je primjer uplatnice koja je skenirana [27]. Nakon uspješnog skeniranja podaci poput opisa, iznosa i tipa transakcije automatski se unose u odgovarajuća polja kao što je vidljivo na slici 5.22. Korisnik tada može pritisnuti gumb „*ADD DATA*“, čime se nova transakcija sprema u sustav, a kružni grafikon se osvježava kako bi prikazao novo unesene podatke.


***Slika 5.21.*** *Primjer uplatnice [27]*


***Slika 5.22.*** *Podaci skenirane uplatnice*

Slika 5.23. prikazuje račun iz kafića, dok slika 5.24. prikazuje već spomenuto automatsko popunjavanje podataka u odgovarajuća polja putem QR *kôda*.


***Slika 5.23.*** *Primjer računa kafića*


***Slika 5.24.*** *Podaci skeniranog računa kafića*

Slika 5.25. prikazuje funkciju „showCamera()“ koja koristi biblioteku ZXing [28] za skeniranje QR *kôdova* u Android aplikaciji. U funkciji se postavljaju različite opcije skeniranja poput vrste *barkôda* (npr. QR *kôdovi*), kamere koja će se koristiti, onemogućavanja zvučnog signala te spremanja slike skeniranog *kôda*. Korisniku se prikazuje prilagođena poruka „*Scan QR Code*“, a orijentacija zaslona nije zaključana, čime se omogućuje skeniranje u portretnom ili pejzažnom načinu rada. Na kraju se pokreće kamera s definiranim opcijama putem objekta „scanLauncher“.

```
private fun showCamera() {
    val options = ScanOptions().apply { this: ScanOptions
        setDesiredBarcodeFormats(ScanOptions.ALL_CODE_TYPES)
        setPrompt("Scan QR Code")
        setCameraId(0)
        setBeepEnabled(false)
        setBarcodeImageEnabled(true)
        setOrientationLocked(false)
    }
    scanLauncher.launch(options)
}
```

***Slika 5.25.*** *Prikaz skeniranja QR kôda putem kamere*

Slika 5.26. prikazuje funkciju scanLauncher() koja pokreće aktivnost skeniranja QR *kôda* u Android aplikaciji. Nakon što je skeniranje završeno, funkcija provjerava sadržaj rezultata. U slučaju da je skeniranje otkazano prikazuje se poruka „*Cancelle*d!“. Ako je skeniranje uspješno tekst se razdvaja i unosi u varijablu receipt, a daljnja obrada ovisi o vrsti skeniranog dokumenta.

Ako je riječ o računu iz kafića funkcija prepoznaje taj obrazac te automatski unosi vrstu unosa kao „Račun iz kafića“ i iz izvučenog sadržaja preuzima iznos računa koji se upisuje u odgovarajuće polje.

Ako se skenira uplatnica ili neki drugi tip dokumenta izvršava se alternativna logika u bloku *else*, u kojem se vrsta unosa i iznos popunjavaju iz drugih dijelova skeniranog sadržaja. Na kraju, bez obzira na vrstu dokumenta, opis se automatski postavlja na zadanu vrijednost iz resursa aplikacije „R.string.expense“.

```
private val scanLauncher = registerForActivityResult(ScanContract()) { it: ScanIntentResult!
    if (it.contents == null) {
        makeToast( message: "Cancelled!")
    } else {
        val receipt = it.contents.lines()
        with(binding) { this: FragmentRevenueAndExpenseBinding
            if (receipt.size == 1) {
                binding.textInputEditTextType.setText(receiptText)
                binding.textInputEditTextAmount.setText(
                    extractValueFromReceipt(
                        it.contents.substringAfter(
                            delimiter: "izn="
                        )
                    )
                )
            } else {
                textInputEditTextType.setText(receipt[13])
                textInputEditTextAmount.setText(extractValueFromReceipt(receipt[2]))
            }
            textInputEditTextDescription.setText(getString(R.string.expense))
        }
    }
}
```

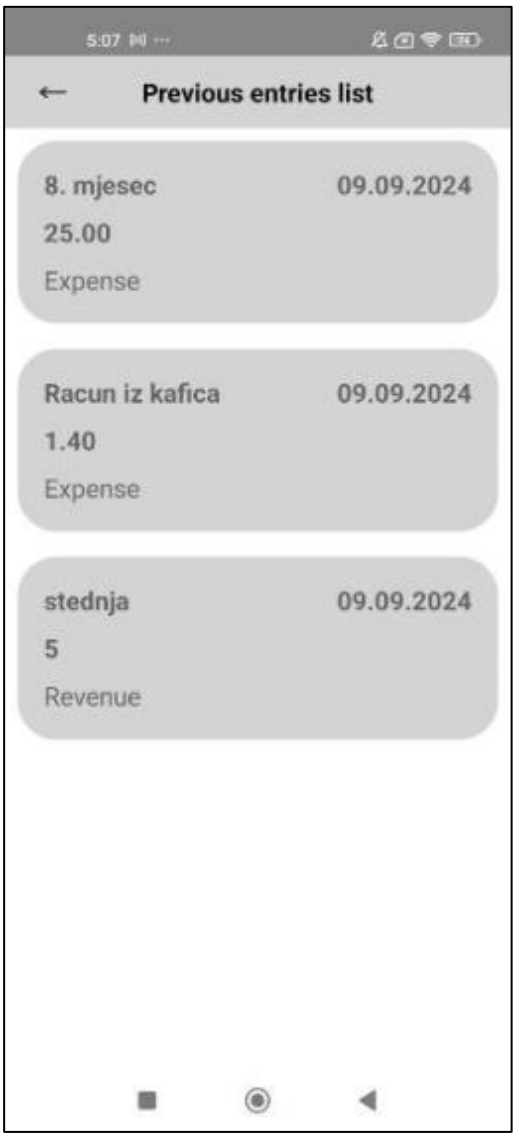
***Slika 5.26.*** *Prikaz pokretanja skeniranja QR kôda*

## 5.4. Pregled dodanih prihoda/rashoda

Slika 5.27. prikazuje zaslon s popisom prethodnih unosa, a koji obuhvaćaju sve prihode i rashode korisnika. Na vrhu zaslona nalazi se traka s nazivom stranice „*Previous entries list*“, a s lijeve strane nalazi se ikona koja korisniku omogućuje povratak na prethodni zaslon. Ispod trake s nazivom stranice prikazan je popis svih prihoda i rashoda u obliku kartica. Svaka kartica prikazuje ključne detalje za pojedini unos, uključujući:

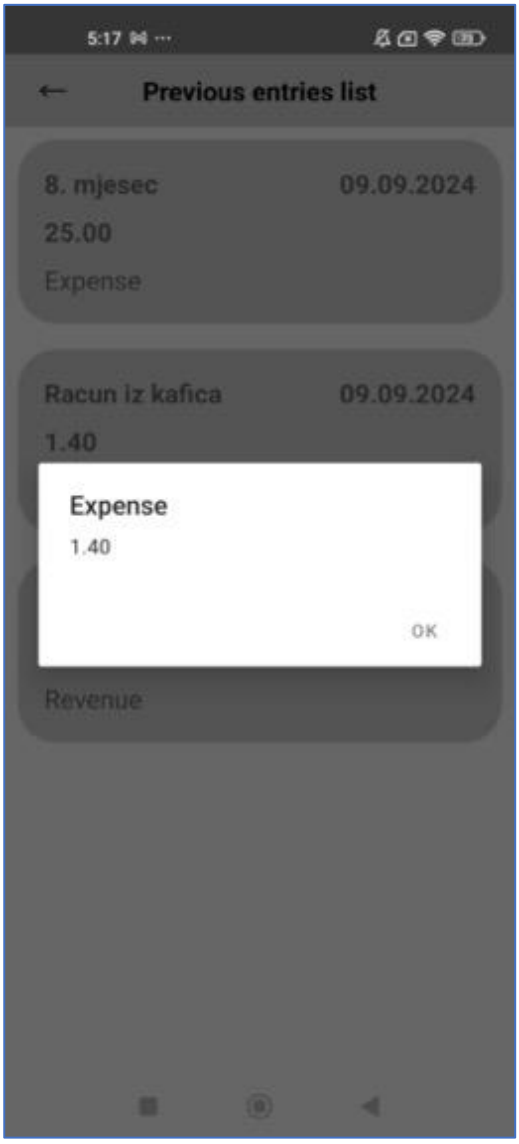
- Naziv unosa (npr. „Racun iz kafica“)
- Iznos transakcije (npr. „1.40“)
- Vrstu transakcije (prihod „*Revenue*“ ili rashod „*Expense*“)
- Datum transakcije (npr. „09.09.2024“)

Zaslon omogućuje korisnicima brz pregled i razlikovanje pojedinih unosa putem jasno strukturiranih informacija na svakoj kartici.




***Slika 5.27.*** *Prikaz popisa prethodnih unosa s detaljima transakcija*

Slika 5.28. prikazuje situaciju nakon što korisnik klikne na jedan od elemenata u listi. U ovom slučaju otvara se skočni dijalog koji sadrži više informacija o odabranom unosu. Dijalog prikazuje detalje o vrsti transakcije (npr. „*Expense*“) i iznosu transakcije (npr. „1.40“). Korisnik zatim može potvrditi zatvaranje dijaloga klikom na gumb „*OK*“. Ovaj dijalog omogućava korisnicima da brzo dobiju dodatne informacije o transakciji bez napuštanja glavnog zaslona s popisom unosa.




***Slika 5.28.*** *Prikaz detalja unosa iz popisa prethodnih transakcija*

# 6. ZAKLJUČAK

U ovom diplomskom radu uspješno je razvijena i implementirana mobilna Android aplikacija za upravljanje osobnim financijama koja je osmišljena s naglaskom na intuitivno korisničko iskustvo i bogatu funkcionalnost. Detaljno su opisane korištene tehnologije, alati te arhitektura aplikacije, pri čemu su istaknute ključne značajke koje korisnicima omogućuju jednostavno evidentiranje prihoda i rashoda. Prikazani su najvažniji dijelovi korisničkog sučelja i mogući scenariji korištenja aplikacije, čime je osigurana preglednost i jasna funkcionalnost. Uz vizualni pregled temeljito su objašnjeni ključni programski moduli koji omogućuju unos, pregled i kategorizaciju financijskih podataka.

Korištenjem naprednih tehnologija poput OCR-a za prepoznavanje računa i integracije QR *kôdova* aplikacija omogućuje brzo i precizno unošenje financijskih transakcija značajno smanjujući potrebu za ručnim unosom podataka. Razvoj je izveden prema najmodernijim arhitekturalnim obrascima, čime je omogućeno lako proširenje aplikacije, jednostavno testiranje i održavanje uz naglasak na modularnost i skalabilnost sustava, što osigurava dugoročnu održivost projekta.

Aplikacija je uspješno realizirana i u potpunosti ispunjava sve postavljene zahtjeve. Unatoč tome postoji prostor za daljnje poboljšanje i proširenje funkcionalnosti. Moguća nadogradnja uključuje uvođenje automatske analize troškova te vizualizaciju financijskih podataka putem grafova, što bi korisnicima omogućilo dublji uvid u njihove financijske navike.

Dodatno, implementacija vlastitog poslužitelja mogla bi povećati sigurnost i fleksibilnost aplikacije pružajući širi spektar mogućnosti u odnosu na trenutnu integraciju s Firebaseom. Unatoč tim potencijalnim unaprjeđenjima, aplikacija već sada pruža stabilne i pouzdane funkcionalnosti uspješno zadovoljavajući sve postavljene zahtjeve i ciljeve, omogućujući korisnicima da na jednostavan i učinkovit način organiziraju svoje financije – što je i bio glavni cilj ovog diplomskog rada.

# LITERATURA


[1] „8 benefits of financial apps“ [online]. Dostupno na: https://www.securesave.com/blog/8-benefits-of-financial-apps. [Pristupljeno: 22.6.2024.].

[2] D., French, D., McKillop, E., Stewart, „Personal finance apps and low-income households“, *Strategic Change*, sv. 30, str. 367–375, srp. 2021.

[3] R., Deb, „Influence of Mobile Apps on Household Saving-Spending Behaviour“, izd. 2, 2020.

[4] „Goodbudget: Budget & Finance - Apps on Google Play“ [online]. Dostupno na: https://play.google.com/store/apps/details?id=com.dayspringtech.envelopes&hl=en. [Pristupljeno: 28.6.2024.].

[5] „Our App Lineup“ [online]. Dostupno na: https://www.ynab.com/our-app-lineup. [Pristupljeno: 28.6.2024.].

[6] „Ynab picture.png (1538×1553)“ [online]. Dostupno na: https://cdn.prod.website-files.com/640f69143ec11b21d42015c6/6502e96f8b7ff92feac8c8ab_hero-phone.png. [Pristupljeno: 28.6.2024.].

[7] „What is Android Studio?“ [online], 07-tra-2023. Dostupno na: https://www.geeksforgeeks.org/overview-of-android-studio/. [Pristupljeno: 26.6.2024.].

[8] „Koin - The Kotlin Dependency Injection Framework“ [online]. Dostupno na: https://insert-koin.io/. [Pristupljeno: 27.6.2024.].

[9] „Koin Basics“ [online]. Dostupno na: https://hyperskill.org/learn/step/40052. [Pristupljeno: 27.6.2024.].

[10] „What is Android Studio? | Definition from TechTarget“ [online]. Dostupno na: https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio. [Pristupljeno: 26.6.2024.].

[11] „Kotlin overview“ [online]. Dostupno na: https://developer.android.com/kotlin/overview. [Pristupljeno: 26.6.2024.].

[12] M., Heller, „What is Kotlin? The Java alternative explained“ [online], 14-lis-2022. Dostupno na: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html. [Pristupljeno: 26.6.2024.].

[13] „Getting to Grips with MVVM Architecture“ [online]. Dostupno na: https://www.netguru.com/blog/mvvm-architecture. [Pristupljeno: 26.6.2024.].

[14] „Uvod u MVVM arhitekturu“ [online], 31-srp-2019. Dostupno na: https://www.webprogramiranje.org/uvod-u-mvvm-arhitekturu/. [Pristupljeno: 26.6.2024.].

[15] N., Nikitins, „Understanding the Model-View-ViewModel (MVVM) Pattern: A Guide for Software Developers“ [online], 11-ožu-2023. .

[16] „What is Firebase?“ [online]. Dostupno na: https://www.educative.io/answers/what-is-firebase. [Pristupljeno: 26.6.2024.].

[17] D., Stevenson, „What is Firebase? The complete story, abridged.“ [online], 25-lis-2018. .

[18] „Coroutines | Kotlin“ [online]. Dostupno na: https://kotlinlang.org/docs/coroutines-overview.html. [Pristupljeno: 27.6.2024.].

[19] „Save data in a local database using Room“ [online]. Dostupno na: https://developer.android.com/training/data-storage/room. [Pristupljeno: 27.6.2024.].

[20] „Define data using Room entities“ [online]. Dostupno na: https://developer.android.com/training/data-storage/room/defining-data. [Pristupljeno: 27.6.2024.].

[21] „Overview of Room in Android Architecture Components“ [online], 05-svi-2021.

Dostupno na: https://www.geeksforgeeks.org/overview-of-room-in-android-architecture-components/. [Pristupljeno: 27.6.2024.].
[22] „Dependency injection with Kotlin using the Koin library – Ona“ [online]. .
[23] „Navigation“ [online]. Dostupno na: https://developer.android.com/guide/navigation. [Pristupljeno: 28.6.2024.].
[24] ikust, „The Navigation Architecture Component Tutorial: Getting Started“ [online]. Dostupno na: https://www.kodeco.com/6014-the-navigation-architecture-component-tutorial-getting-started. [Pristupljeno: 28.6.2024.].
[25] „What is a Flowchart? Process Flow Diagrams & Maps | ASQ“ [online]. Dostupno na: https://asq.org/quality-resources/flowchart. [Pristupljeno: 28.6.2024.].
[26] E., Dangerfield, „Tutorial: Modern Android Development with MVVM, LiveData and Firebase (Part 2)“ [online], 05-ožu-2020. .
[27] „Generator naloga za nacionalna plaćanja HUB 3A“ [online]. Dostupno na: https://knee-cola.github.io/generator-opce-uplatnice/. [Pristupljeno: 9.9.2024.].
[28] „QR_OCR_Code - journeyapps/zxing-android-embedded“. JourneyApps, 11-ruj-2024.

# SAŽETAK

U ovom radu prikazana je važnost praćenja osobnih financija putem mobilnih aplikacija koje pojednostavljuju upravljanje prihodima i rashodima. Za potrebe ovog diplomskog rada izrađena je mobilna aplikacija za Android platformu, osmišljena kako bi korisnicima omogućila jednostavno evidentiranje financijskih podataka. Aplikacija koristi napredne tehnologije poput OCR-a za automatsko prepoznavanje računa i QR *kôdova*, što omogućuje brz i točan unos podataka. Također, aplikacija podržava kategorizaciju troškova po različitim stavkama, čime korisnici dobivaju bolji uvid u svoju potrošnju. Aplikacija je razvijena u Kotlin programskom jeziku unutar Android Studio okruženja, a koristi se modularna arhitektura koja omogućuje skalabilnost, jednostavno održavanje i buduća proširenja.

Ključne riječi: Kotlin, osobne financije, QR *kôd*

## ABSTRACT

This paper presents the importance of tracking personal finances through mobile applications that simplify the management of income and expenses. For the purposes of this thesis, a mobile application for the Android platform was developed, designed to enable users to easily record financial data. The application uses advanced technologies such as OCR for automatic receipt recognition and QR codes, allowing for fast and accurate data entry. Additionally, the application supports the categorization of expenses into various categories, giving users better insight into their spending. The application was developed in the Kotlin programming language within the Android Studio environment, utilizing a modular architecture that allows for scalability, easy maintenance, and future upgrades.



Keywords: Kotlin, personal finances, QR code

# ŽIVOTOPIS

Benjamin Čaušić rođen je 13. lipnja 1997. godine u Osijeku, a trenutno živi u Valpovu. Osnovno obrazovanje stekao je u Osnovnoj školi Matije Petra Katančića u Valpovu, dok je srednjoškolsko obrazovanje završio u Srednjoj školi Valpovo. Trenutno je student zadnje godine diplomskog sveučilišnog studija računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Tijekom studija aktivno je usmjeren na razvoj svojih vještina u struci te je započeo profesionalnu karijeru kao inženjer za testiranje softvera, čime dodatno unapređuje svoje tehničko znanje u području testiranja i razvoja aplikacija.

# PRILOZI

Prilog 1. Diplomski rad u formatu .docx

Prilog 2. Diplomski rad u formatu .pdf

Prilog 3. GitHub repozitorij programskog *kôda* aplikacije [On-line] dostupan je na linku: https://github.com/causicb/WealthWise_BenjaminCausic