Mobilna aplikacija za upravljanje osobnim financijama

Marčec, Filip

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 424

MOBILNA APLIKACIJA ZA UPRAVLJANJE OSOBNIM FINANCIJAMA

Filip Marčec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 424

MOBILNA APLIKACIJA ZA UPRAVLJANJE OSOBNIM FINANCIJAMA

Filip Marčec

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 424

Pristupnik:	**Filip Marčec (0036525149)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	prof. dr. sc. Krešimir Fertalj

Zadatak: **Mobilna aplikacija za upravljanje osobnim financijama**

Opis zadatka:

Proučiti nekoliko reprezentativnih aplikacija za praćenje osobnih troškova. Definirati skup najčešće zastupljenih funkcija te napraviti specifikaciju zahtjeva na vlastito rješenje. Projektirati i ugraditi odgovarajuću aplikaciju nad bazom podataka. Aplikacija treba omogućiti definiranje vlastitih kategorija prihoda i rashoda. Omogućiti evidentiranje ponavljajućih prihoda i rashoda po kategorijama. Ugraditi statističku obradu povijesnih podataka uz vizualizaciju po različitim vrstama grafičkog prikaza. Automatizirati postupak unosa podataka.

Rok za predaju rada: 28. lipnja 2024.

SADRŽAJ

Uvod

Upravljanje novcem i osobnim financijama važan je aspekt našeg svakodnevnog života. U današnjem ubrzanom svijetu, lakše je nego ikad izgubiti pregled nad navikama trošenja, što može dovesti do financijskih problema. Nažalost, svakodnevna je pojava gdje se ljudi zbog nedostatka razumijevanja gdje njihov novac odlazi suočavaju s financijskim poteškoćama. Bilo da je riječ o želji za ostvarivanjem štednje, otplaćivanjem već postojećih dugova ili da samo žele uspostaviti život i potrošačke navike koje su u skladu s vlastitim mogućnostima, praćenje troškova i upravljanje financijama početak su rješenja navedenih scenarija.

Jedan od temeljnih koraka koje je potrebno poduzeti je kategorizacija troškova. Razvrstavanjem troškova u određene kategorije, poput osnovnih životnih potreba, zdravstva, obrazovanja, zabave i štednje, dobiva se jasnija slika o tome gdje novac odlazi. Kategorizacija troškova omogućuje identifikaciju stvarno potrebnih troškova, kao što su hrana, stanovanje i zdravstveno osiguranje, od onih koje je moguće smanjiti ili čak potpuno izbaciti, poput luksuznih artikala, pretjerane potrošnje na zabavu ili nepotrebne pretplate. Sljedeći korak je postavljanje financijskih ciljeva. Jasno definirani ciljevi pružaju motivaciju i smjer za upravljanje financijama. Analizom prošlih troškova mogu se uočiti trendovi u potrošnji, identificirati periode povećane potrošnje i prepoznati područja gdje je moguće primijeniti dodatne uštede. Ovaj uvid omogućuje prilagodbu financijskih navika kako bi se izbjeglo ponavljanje istih grešaka.

Upravljanje financijama može se obavljati na više načina, a izbor metode ovisi o osobnim preferencijama i stilu života pojedinca. Jedan od najjednostavnijih pristupa je ručno vođenje bilješki gdje se zapisuje svaka transakcija. Ovaj metodički pristup ne samo da pomaže u praćenju troškova, već i potiče svijest o financijskim navikama. S druge strane, Excel tablica je popularan izbor za one koji preferiraju digitalni format. Excel omogućava stvaranje detaljnih proračuna s mogućnošću korištenja formula i grafikona za bolju analizu podataka. Ova metoda omogućuje lakše praćenje trendova i usporedbu izdataka iz mjeseca u mjesec. Excel tablice su fleksibilne i mogu se prilagoditi specifičnim potrebama korisnika, što ih čini izvrsnom opcijom za detaljno upravljanje financijama. Obje metode zahtijevaju određenu razinu posvećenosti i konzistentnosti u vođenju zapisnika i praćenju troškova, što može biti izazov za neke ljude. U svakom slučaju, bitno je pronaći ravnotežu između detaljnosti podataka i praktičnosti u svakodnevnoj upotrebi kako bi upravljanje financijama bilo održivo i korisno.

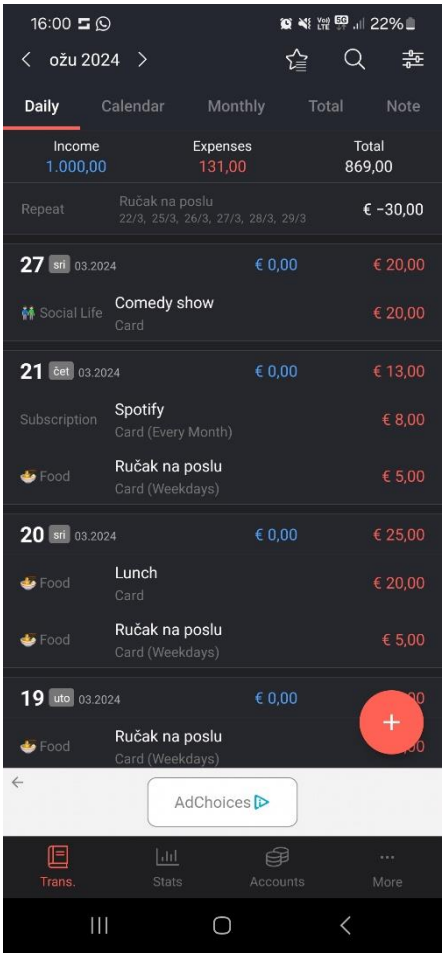
Kao odgovor na spomenute probleme nameću se mobilne aplikacije za upravljanje financijama. Ove aplikacije nude niz funkcionalnosti, brz unos transakcija, pristupačna kategorizacija transakcija, automatski prikaz podataka u raznim grafikonima, definiranje ponavljajućih transakcija, postavljanje financijskih ciljeva i slično.

Cilj ovog diplomskog rada je razviti mobilnu aplikaciju koja će korisnicima omogućiti učinkovito upravljanje njihovim osobnim financijama. Aplikacija će nuditi korištenja s funkcije koje olakšavaju svakodnevno praćenje troškova, kategorizaciju transakcija i vizualizaciju financijskih podataka putem grafikona. U nastavku će biti analizirane dvije reprezentativne mobilne aplikacije kako bi se identificirale funkcionalnosti koje se najčešće pojavljuju i eventualni nedostaci.

1. Analiza postojećih rješenja za odabrano područje

1.1. Money Manager Expense & Budget

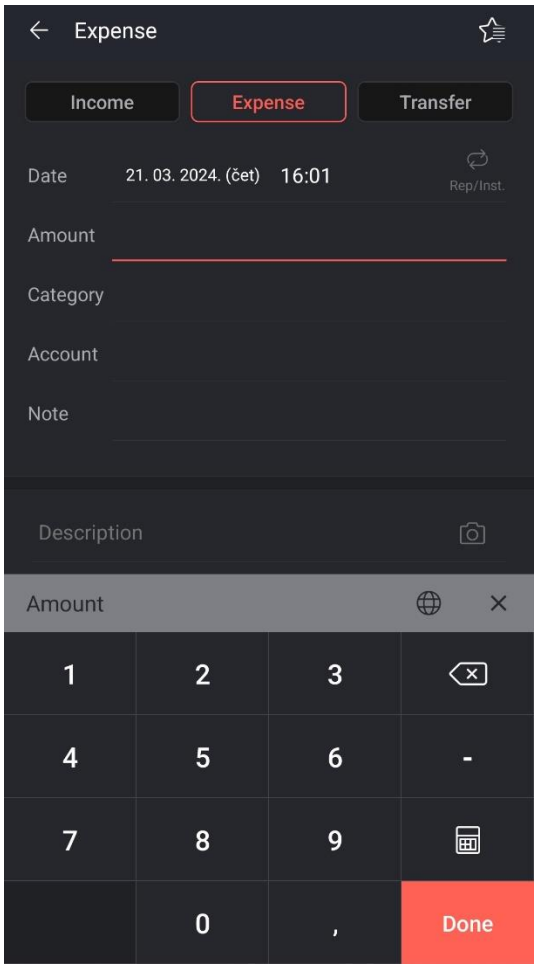
Money Manager Expense & Budget [1] je mobilna aplikacija dizajnirana za pomoć u upravljanju osobnim financijama s ciljem pojednostavnjenja procesa praćenja troškova i izrade proračuna. Razvijena od strane Realbyte Inc., aplikacija omogućava korisnicima da učinkovito evidentiraju svoje dnevne transakcije, vizualiziraju financijske trendove kroz razne izvještaje i grafikone te planiraju buduće troškove kako bi ostvarili bolju financijsku disciplinu.



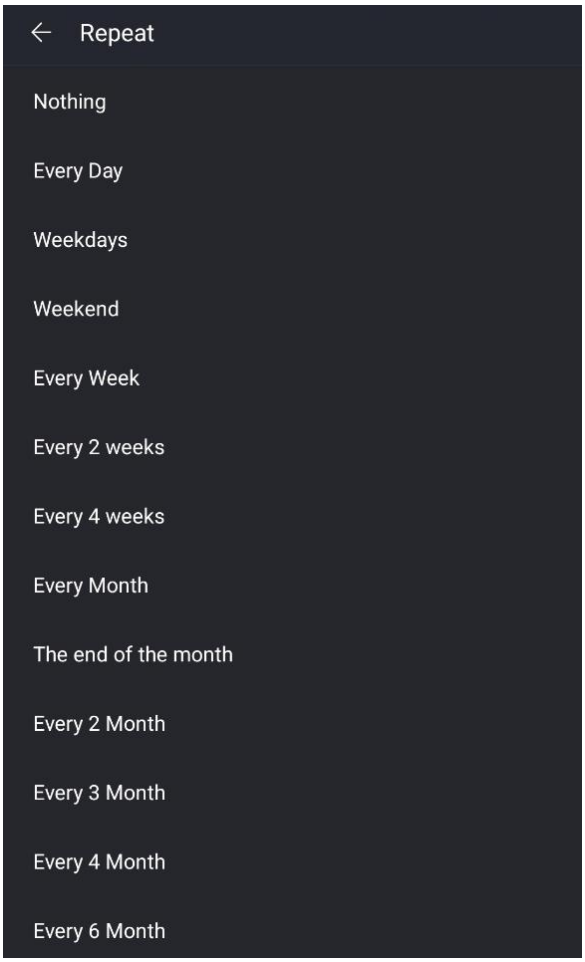
Slika 1. Početni zaslon

Na slici 1. nalazi se početni zaslon aplikacije. U donjem dijelu zaslona nalazi se navigacijska traka s četiri opcije : transakcije, statistika, računi i postavke. U gornjem dijelu zaslona nalaze se kartice koje omogućuju pregled podataka po danima, mjesecima, ukupno te pregled bilješki koje korisnik može generirati. Ispod komponente s prethodno navedenim karticama nalazi se komponenta u kojoj su naznačeni ukupni prihodi, troškovi i ukupno stanje računa. U gornjem

desnom kutu nalazi se komponenta s tri dugmeta. Jedan služi za pregled troškova ili prihoda koji su označeni kao „predložak“ radi bržeg unosa ukoliko je riječ o zapisu koji se ponavlja. Tu je još dugme za pretraživanje i dugme za postavke. U lijevom kutu također se nalazi i dugme koje služi za promjenu mjeseca i godine za koje se prikazuju podaci.



Slika 3. Unos prihoda/troškova

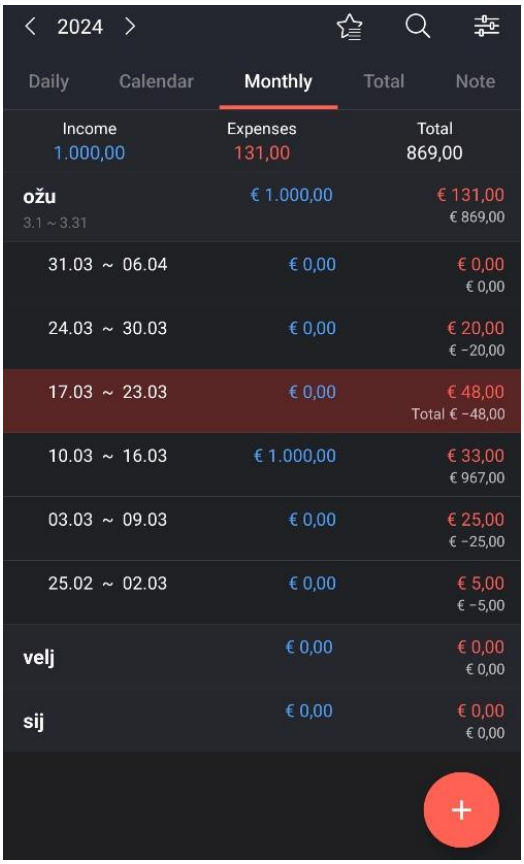


Slika 2. Opcije za odabir ponavljanja

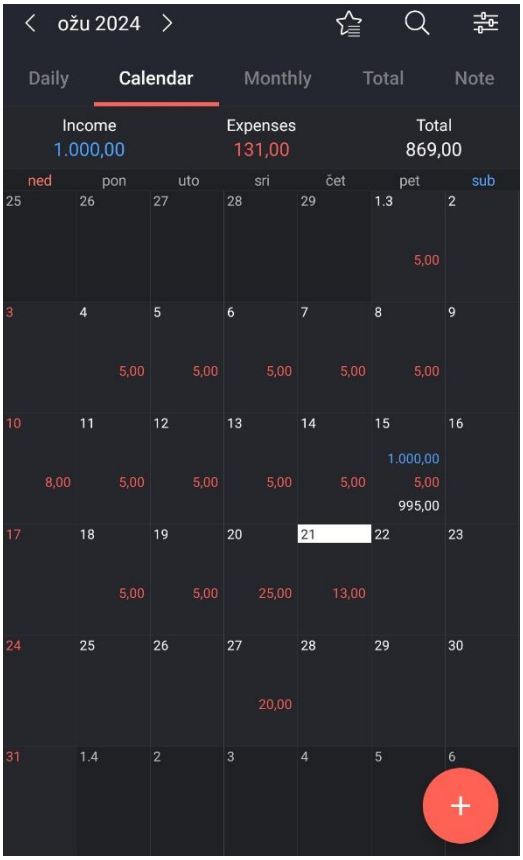
U središnjem dijelu zaslona nalazi se po datumima sortiran popis troškova i uplata koji je grupiran po danima jer se aplikacija trenutno nalazi u dnevnom prikazu. Svaki od dana prezentiran je u obliku jedne kartice koja u vrhu plavom bojom ima zbroj svih prihoda, a zbroj svih troškova je označen crvenom bojom. Zatim, svaka od stavki koja može biti trošak ili prihod, ima prikazanu kategoriju kojoj pripada, kratak naziv i način plaćanja ukoliko je riječ o trošku.

Na slici 2. nalazi se prikaz zaslona za unos novog zapisa – trošak, prihod ili transfer, kojem se pristupa pritiskom na crveno dugme sa znakom plus koje se nalazi u donjem desnom dijelu zaslona na slici 1. U ovoj formi moguće je odabrati je li riječ u prihodu, trošku ili prijenosu novca s jednog računa na drugi. Zatim je potrebno upisati datum te odabrati opciju koja

omogućuje postavljanje ponavljajućeg troška ili prihoda, ukoliko je o takvom riječ. Na slici 3. moguće je vidjeti sve opcije koje se nude za odabir kod postavljanja ponavljajućeg troška ili prihoda. Potrebno je upisati brojčani iznos troška/prihoda, odabrati kategoriju, odabrati s kojeg računa će se iznos oduzeti, odnosno dodati ako je riječ o prihodu. Za kraj je moguće dodati kratak opis zapisa.



Slika 5. Prikaz "Monthly"



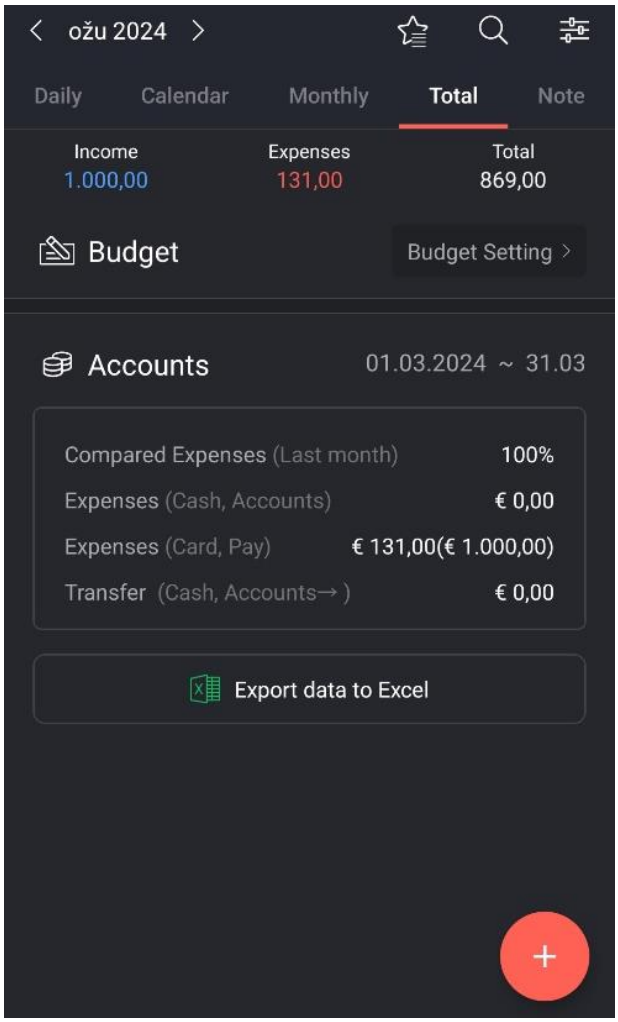
Slika 4. Prikaz kalendara

Na slici 4. nalazi se prikaz koji se dobije pritiskom na dugme „Calendar“ u gornjem dijelu ekrana. U gornjem dijelu ekrana ostaje isti prikaz kao i na slici 1., komponenta u kojoj su naznačeni ukupni prihodi, troškovi i ukupno stanje računa. U središnjem dijelu nalazi se kalendarski prikaz mjeseca gdje je za svaki dan crvenom bojom označen ukupan dnevni trošak, a plavom bojom eventualni prihodi. Svaki od dana je moguće dodirnuti te pregledati opširniji popis troškova i prihoda.

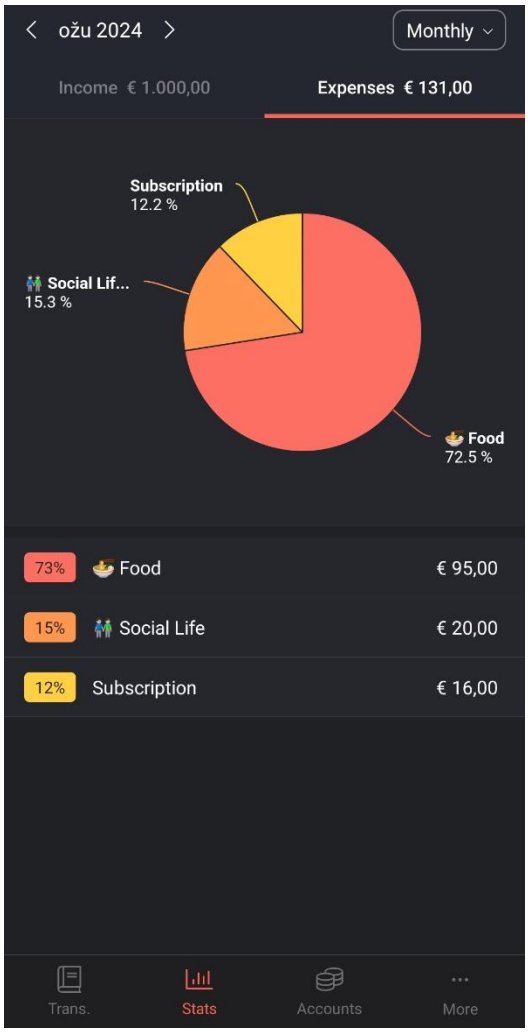
Na slici 5. nalazi se prikaz kojem se može pristupiti klikom na dugme „Monthly“. U ovom prikazu nalazi se popis mjeseci za trenutno odabranu godinu koju je moguće promijeniti u gornjem lijevom kutu zaslona. Odabirom pojedinog mjeseca nudi se prikaz po tjednima gdje

je za svaki tjedan naznačen ukupan trošak i ukupni prihodi. Odabirom na neki od tjedana moguće je pregledati opširniji popis zapisa.

Na slici 6. moguće je vidjeti zaslon „Total“ gdje se nalazi prikaz ukupnih troškova i prihoda za odabrani mjesec. Na ovom zaslonu se nudi opcija postavljanja mjesečnog budžeta, kao i opcija izvoza podataka u Excel.



Slika 6. Prikaz "Total"



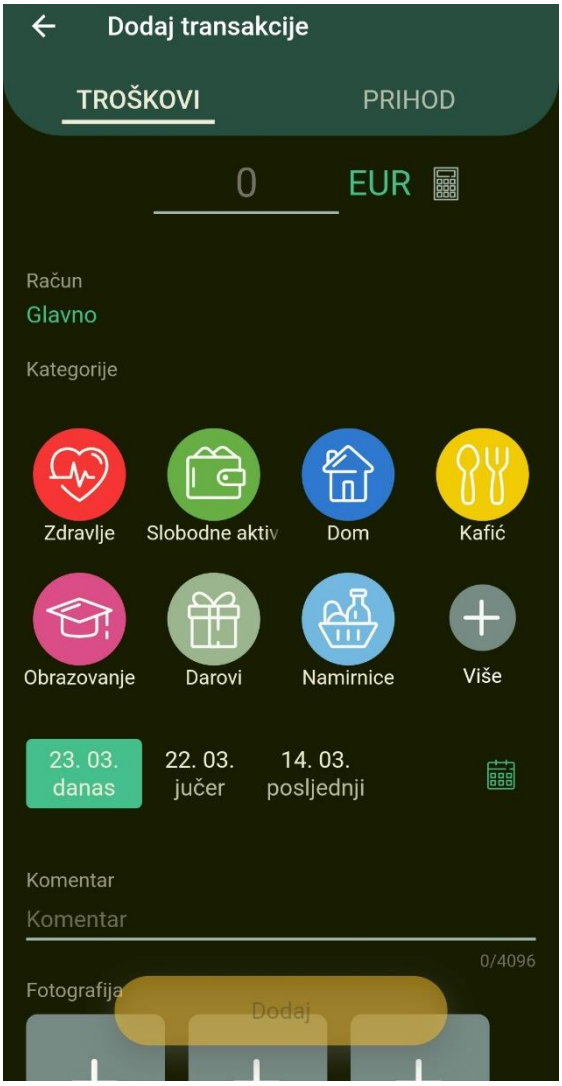
Slika 7. Prikaz statistike

Na slici 7. nalazi se prikaz kojem se pristupa pritiskom na dugme „Stats“ u donjem dijelu zaslona. Ovaj zaslona nudi prikazane podatke o troškovima ili prihodima u obliku kružnog dijagrama, gdje su različitim boja označene različite kategorije kojima troškovi ili prihodi pripadaju. Ispod kružnog dijagrama nalazi se popis svih prikazanih kategorija koji je moguće filtrirati. U ovom prikazu također je moguće odabrati mjesečni, tjedni ili godišnji prikaz. Aplikacija još nudi opciju stvaranja vlastitih kategorija troškova i prihoda, što povećava mogućnost personalizacije iskustva korištenja.

1.2. Financije – upravitelj novca

Drugo postojeće rješenje jest mobilna aplikacija pod nazivom „Financije – upravitelj novcem“ [2]. Aplikacija nudi standardne očekivane funkcionalnosti poput unosa transakcija koje mogu biti trošak ili prihod, odabir kategorije transakcije, dodavanje vlastite kategorije, mogućnost postavljanja ponavljajućih troškova i slično. U nastavku slijedi opširniji pregled spomenutih funkcija uz priložene snimke zaslona iz aplikacije.

Slika 8. "Financije - upravitelj novcem" - početni zaslon

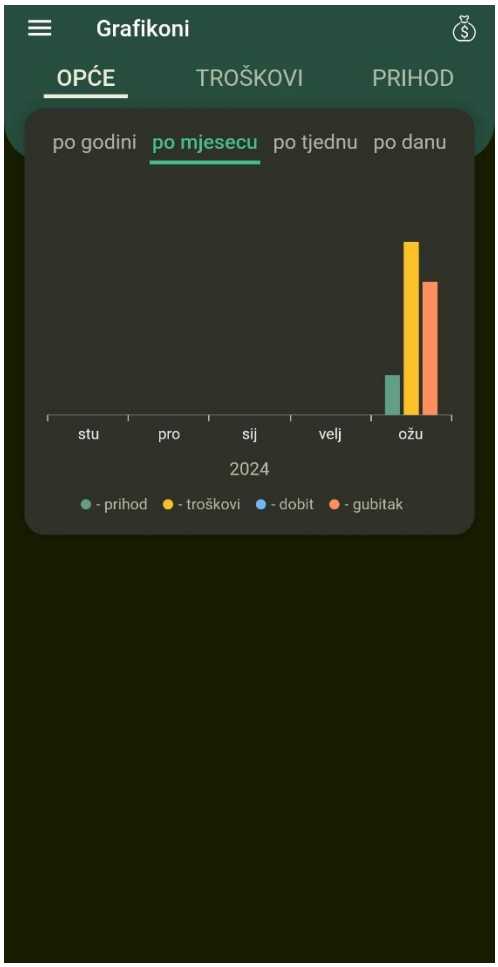


Slika 9. "Financije - upravitelj novcem" - dodavanje transakcije

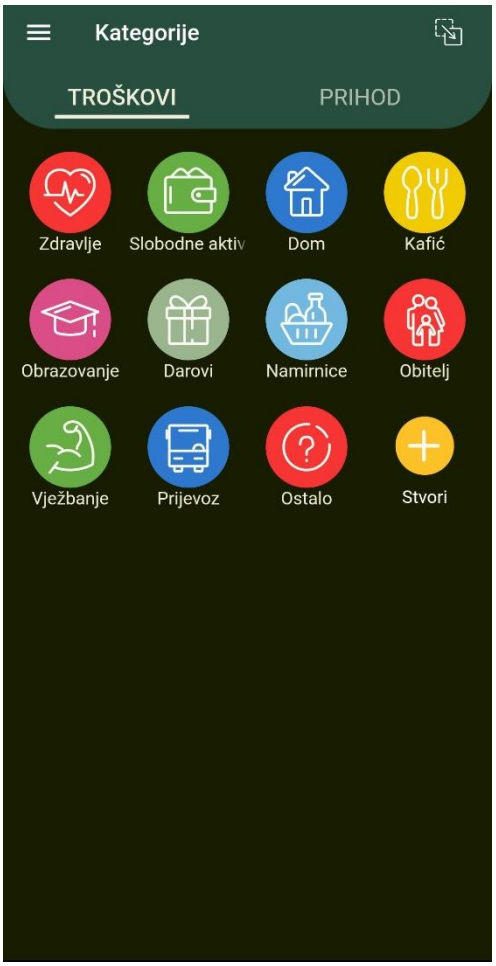
Početni zaslon aplikacije prikazuje kružni dijagram koji prikazuje kategorizirane troškove i njihov ukupan iznos. Desno od dijagram nalazi se dugme koje omogućuje dodavanje nove transakcije. U ovom prikazu moguće je pregledati ukupne troškove ili prihode za odabrano vremensko razdoblje – dan, tjedan, mjesec, godina ili specifično odabrano razdoblje. U donjoj

polovici zaslona nalazi se popis transakcije po kategorijama u koje pripadaju i njihov udio u ukupnom iznosu za odabrano razdoblje.

Na slici 9. nalazi se forma kojom se dodaje nova transakcija. Potrebno je odabrati račun za koji se nova transakcija bilježi, upisati iznos u odabranoj valuti, odabrati kategoriju iz skupa unaprijed definiranih kategorija ili stvoriti vlastitu kategoriju. Zatim je potrebno odabrati datum izvršavanja transakcije, a moguće je ostaviti i komentar ili transakciji pridijeliti fotografije.

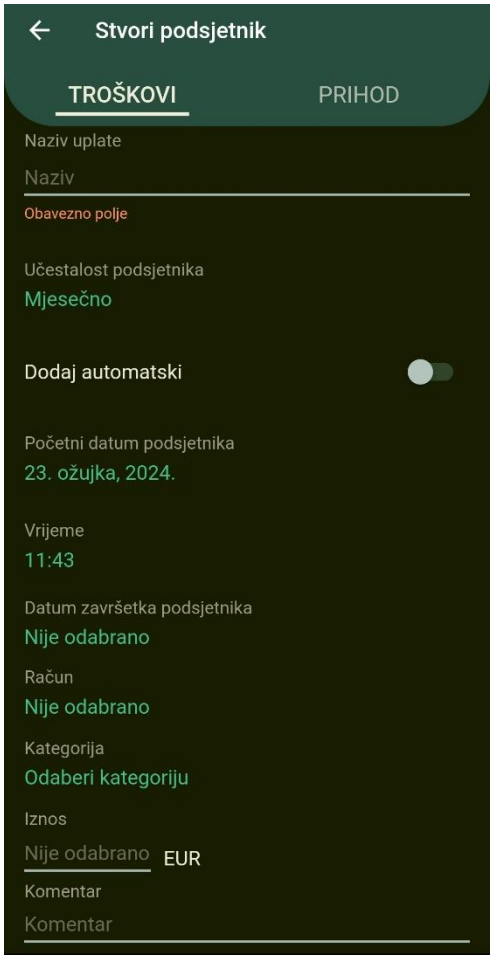


Slika 10. "Financije - upravitelj novcem" - grafikoni

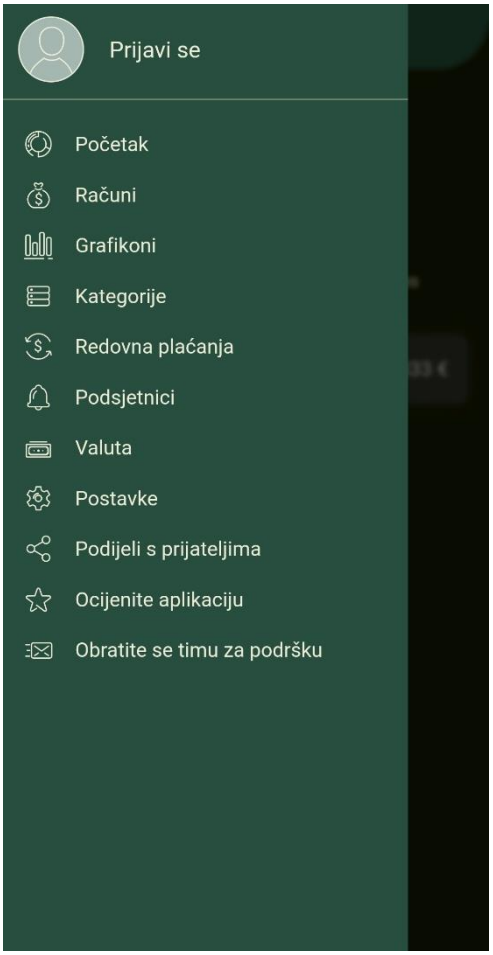


Slika 11. "Financije - upravitelj novcem" – upravljanje kategorijama

Na slici 10. nalazi se grafikoni koji nudi tri tipa prikaza – opći, samo troškovi i samo prihodi. Riječ je o stupčastom dijagramu koji za svaki mjesec u tekućoj godini prikazuje relativne iznose prihoda, troškova, dobiti i gubitka. Ovdje je također moguće odabrati prikaz po godini, tjednu ili danu. Na slici 11. vidljive su unaprijed definirane kategorije koje aplikacija nudi, a također postoji mogućnost stvaranja vlastite kategorije.



Slika 12. "Financije - upravitelj novcem" – stvaranje podsjetnika



Slika 13. "Financije - upravitelj novcem" – izbornik

Ova aplikacija također nudi stvaranje ponavljajućih transakcija koje je moguće stvoriti da se ponavljaju svaki odabrani vremenski period kako bi se automatiziralo praćenje financija. Forma koja to omogućava vidljiva je na slici 12. Na slici 13. vidljiv je izbornik u kojem je moguće dodati različite račune, dodati nove kategorije, promijeniti valutu s kojom aplikacija radi i promijeniti određene vizualne aspekte koji ne utječu na funkcionalnosti koje su prethodno opisane.

1.3. Zajedničke funkcionalnosti analiziranih aplikacija

Iz analize aplikacija "Money Manager Expense & Budget" i "Financije – upravitelj novcem", možemo izdvojiti sljedeće zajedničke funkcionalnosti:

- Evidencija transakcija

Obje aplikacije omogućuju korisnicima da bilježe dnevne transakcije, bilo da su troškovi ili prihodi. Ovo uključuje dodavanje detalja poput iznosa, datuma, kategorije i opcionalno opisa ili fotografija transakcije.

- Kategorizacija transakcija

Korisnici mogu kategorizirati svaku transakciju prema unaprijed definiranim kategorijama koje aplikacije nude ili mogu stvoriti vlastite kategorije za bolju organizaciju i analizu troškova i prihoda.

- Vizualizacija financija

Obje aplikacije nude grafičke prikaze financijskih podataka kroz različite izvještaje i grafikone, kao što su kružni dijagrami, stupčasti dijagrami i kalendarski prikazi. Ovo pomaže u vizualnom predstavljanju financijskih trendova i lakšem uvidu u financijsko stanje.

- Ponavljajuće transakcije

Mogućnost postavljanja ponavljajućih troškova ili prihoda na određene vremenske intervale kako bi se automatiziralo praćenje redovitih financijskih obveza ili prihoda.

- Pregled po vremenskim periodima

Korisnici mogu pregledavati svoje financije za različite vremenske periode, kao što su dnevni, tjedni, mjesečni ili godišnji pregledi, što omogućuje fleksibilno praćenje i analizu financijskih navika.

- Upravljanje više računa

Obje aplikacije omogućuju dodavanje i upravljanje više računa unutar aplikacije, što korisnicima pruža mogućnost da na jednom mjestu imaju pregled svih svojih financija.

- Proračun i ciljevi

Korisnici mogu postaviti mjesečne proračune za različite kategorije ili ukupne financije te pratiti svoj napredak prema postavljenim financijskim ciljevima.

Ove zajedničke funkcionalnosti pružaju korisnicima sredstva za detaljno praćenje, analizu i planiranje osobnih financija, promičući bolju financijsku disciplinu i svijest.

2. Specifikacija zahtjeva

2.1. Poslovni zahtjevi

- Potreba za razvojem mobilne aplikacije koja će omogućiti korisnicima učinkovito upravljanje osobnim financijama i troškovima.
- Potreba za intuitivnim rješenjem koje ne zahtijeva predznanje o financijama, omogućujući korisnicima lako praćenje i analizu njihove potrošnje.
- Potreba za promicanjem financijske odgovornosti i samodiscipline putem vizualizacije troškova i proračuna.

2.2. Korisnički zahtjevi

- Korisnik mora moći izvršiti registraciju ili prijavu u aplikaciju.
- Korisnik mora moći unijeti, uređivati i obrisati transakcije.
- Korisnik mora moći pregledati transakcije za odabrani vremenski period.
- Korisnik mora moći definirati ponavljajuće transakcije.
- Korisnik mora moći stvoriti, uređivati i obrisati vlastite kategorije transakcija.
- Korisnik mora moći pregledati transakcije prikazane grafikonima.
- Korisnik mora moći primati obavijesti koje služe kao podsjetnik za unos transakcija.

2.3. Funkcionalni zahtjevi

- Aplikacija mora omogućiti registraciju prilikom koje je potrebno unijeti ime, prezime, email i lozinku.
- Aplikacija mora omogućiti prijavu prilikom koje je potrebno unijeti email i lozinku uz uvjet da je registracija prethodno izvršena
- Aplikacija mora omogućiti unos transakcija s detaljima kao što su iznos, datum, kategorija, i opcionalni opis, odabir vrste transakcije – trošak ili prihod.
- Aplikacija mora omogućiti uređivanje transakcija omogućujući korisniku da ažurira bilo koji od navedenih detalja unesene transakcije.
- Aplikacija mora omogućiti brisanje transakcija dajući korisniku mogućnost da ukloni transakciju iz svoje financijske evidencije.
- Aplikacija mora pružati funkcionalnost filtriranja i pregleda transakcija po odabranom vremenskom periodu, kao što su dan, tjedan, mjesec ili godina.

- Aplikacija mora omogućiti korisnicima definiranje ponavljajućih transakcija sa zadavanjem periodičnosti (npr. dnevno, tjedno, mjesečno).
- Aplikacija mora omogućiti korisnicima kreiranje, uređivanje i brisanje prilagođenih kategorija transakcija.
- Aplikacija mora omogućiti vizualizaciju transakcija putem grafikona.
- Aplikacija mora omogućiti slanje obavijesti korisnicima kao podsjetnik za redoviti unos, transakcija, s mogućnošću prilagodbe frekvencije podsjetnika.

2.4. Nefunkcionalni zahtjevi

- Aplikacija treba imati intuitivno korisničko sučelje koje omogućava laku navigaciju i upotrebu bez potrebe za detaljnim tehničkim znanjem.
- Aplikacija treba biti dizajnirana s mogućnošću skaliranja i integracije dodatnih funkcionalnosti kako bi se mogla prilagoditi budućim zahtjevima i trendovima.
- Aplikacija treba osigurati brzo i pouzdano funkcioniranje, minimizirajući vrijeme potrebno za učitavanje i obradu podataka, čime se osigurava dobro korisničko iskustvo.

3. Arhitektura sustava

3.1. Arhitekturni obrazac

Izbor arhitekturnog obrasca je bitan jer osigurava strukturiran pristup dizajniranju sustava, omogućujući jasnu organizaciju i koordinaciju različitih komponenti sustava. Korištenjem odgovarajućeg arhitekturnog obrasca može se postići skalabilnost sustava, što omogućava učinkovit odgovor na povećanje opterećenja ili broja korisnika bez gubitka performansi. Održavanje sustava postaje jednostavnije jer su funkcionalnosti jasno razdvojene i modularizirane, čime se olakšava proces nadogradnje i rješavanja problema. Za potrebe ovog diplomskog rada odabran je arhitekturni obrazac temeljen na modelu klijent-poslužitelj. Ovaj model sastoji se od tri sloja: podatkovnog sloja, sloja poslovne logike i korisničkog to jest prezentacijskog sloja. U nastavku je opisan svaki od tih slojeva.

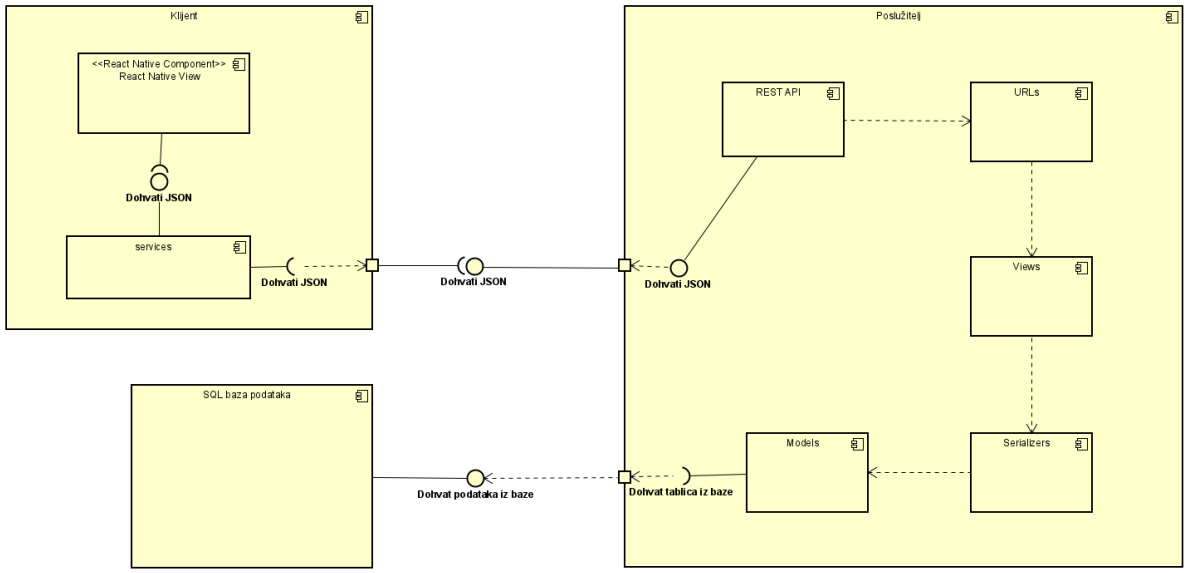
Podatkovni sloj odgovoran je za pohranu i upravljanje podacima sustava. U ovom diplomskom radu koristi se baza podataka PostgreSQL, koja osigurava trajno spremanje podataka, njihovo dohvaćanje, ažuriranje i brisanje. Ovaj sloj omogućava učinkovito rukovanje velikim količinama podataka te uključuje mehanizme za osiguranje integriteta podataka, kontrolu pristupa i zaštitu podataka od neovlaštenih pristupa. Komunikacija s podatkovnim slojem ostvaruje se putem SQL upita ili korištenjem ORM (Object-Relational Mapping) alata unutar razvojnog okvira Django, koji pojednostavljuje rad s bazama podataka.

Sloj poslovne logike implementira poslovna pravila i logiku sustava u aplikaciji razvijenoj u sklopu ovog diplomskog rada. Ovaj sloj obrađuje podatke iz podatkovnog sloja, primjenjuje poslovna pravila, te osigurava da se svi poslovni procesi odvijaju ispravno. Funkcionalnosti sloja poslovne logike mogu uključivati validaciju podataka, proračune, obrade transakcija te orkestraciju komunikacije između različitih dijelova sustava. Korištenje okvira Django omogućava modularnost i održavanje sustava, jer sloj poslovne logike centralizira upravljanje poslovnim pravilima koja se mogu mijenjati bez utjecaja na ostale dijelove aplikacije.

Korisnički ili prezentacijski sloj je onaj dio sustava s kojim krajnji korisnici izravno komuniciraju. U ovom diplomskom radu, ovaj sloj uključuje korisnička sučelja izrađena pomoću React Nativea, koja prikazuju podatke korisnicima i omogućuju im interakciju sa sustavom putem mobilne aplikacije. Prezentacijski sloj odgovoran je za prikupljanje unosa korisnika, slanje zahtjeva sloju poslovne logike, te prikazivanje rezultata korisnicima. U

slučaju mobilne aplikacije, korisnički sloj sastoji se od korisničkog sučelja i omogućuje korisniku interakciju s aplikacijom putem mobilnog uređaja.

3.2. Dijagram komponenti



Slika 14. Dijagram komponenti

Ovaj dijagram prikazuje komponentnu arhitekturu sustava za upravljanje osobnim financijama, koji se sastoji od tri glavne cjeline: klijent, poslužitelj i SQL baza podataka.

Klijent je zadužen za prikaz podataka korisniku i svaki od korisniku vidljivih ekrana zapravo je *React Native view* komponenta koja prima i šalje podatke u JSON formatu. Svaka komponenta poziva metode iz *services* dijela klijenta u kojem su definirane metode za slanje HTTP zahtjeva prema poslužitelju.

Poslužitelj se sastoji od REST API-ja, koji je zadužen za primanje zahtjeva od klijenta i odgovaranje na njih. REST API koristi nekoliko unutarnjih komponenti. Prva komponenta, *URLs*, upravlja usmjeravanjem zahtjeva prema odgovarajućim pogledima (*Views*). Komponenta *Views* obrađuje zahtjeve i priprema podatke za serijalizaciju. *Serializers* komponenta tada pretvara podatke iz modela u JSON format. Na kraju, *Models* predstavlja strukture podataka i definicije tablica u bazi podataka.

SQL baza podataka je komponenta koja čuva sve podatke koje sustav koristi. Poslužitelj dohvaća podatke iz baze putem modela. Modeli predstavljaju strukture podataka i definiraju tablice u bazi podataka. Kada poslužitelj primi zahtjev od klijenta, koristi modele za dohvat

potrebnih podataka iz baze. Baza podataka omogućava pohranu i organizaciju svih transakcija, korisničkih informacija i drugih relevantnih podataka potrebnih za rad sustava.

3.3. Struktura poslužitelja

Struktura repozitorija za ovaj Django projekt organizirana je na način koji osigurava jasnu podjelu odgovornosti i olakšava upravljanje kodom i konfiguracijom. Poslužitelj je sastavljen od dvije glavne aplikacije: *users* i *api*, koje su ujedinjene u glavnom projektu pod nazivom *config*. Ova struktura omogućava modularan i organiziran pristup razvoju i održavanju sustava, tipičan za Django projekte.

Aplikacija *users* pokriva autentifikaciju i upravljanje korisnicima. Koristi prošireni model korisnika *CustomUser* koji omogućava prilagođavanje funkcionalnosti kao što su registracija, prijava, odjava i upravljanje korisničkim profilima. Za autentifikaciju koristi se *JWT* (JSON Web Tokens) implementiran pomoću biblioteke *rest_framework_simplejwt*, što omogućava sigurno upravljanje sjednicama korisnika. Aplikacija *api* zadužena je za pružanje svih ostalih funkcionalnosti sustava. Kroz nju se dohvaćaju i obrađuju podaci potrebni za rad aplikacije, koristeći *Django REST Framework* (DRF) za implementaciju API-ja. Ova aplikacija omogućava kreiranje, čitanje, ažuriranje i brisanje (CRUD) operacija na različitim resursima unutar sustava.

Glavni projekt *config* objedinjuje ove dvije aplikacije i upravlja njihovom interakcijom. U datoteci *urls.py* glavnog projekta definirani su URL obrasci koji usmjeravaju zahtjeve prema odgovarajućim pogledima unutar *users* i *api* aplikacija.

3.3.1. Postavke projekta *settings.py*

U datoteci settings.py definirane su osnovne postavke projekta. Navedene su instalirane aplikacije, *middleware* komponente, konfiguracija baze podataka koristeći *psycopg2* za *PostgreSQL*, postavke za validaciju lozinki, internacionalizaciju te postavke za statičke datoteke. *Middleware* komponente definirane su u *MIDDLEWARE* postavci i uključuju sigurnosne postavke, upravljanje korisničkim sjednicama, CSRF zaštitu i autentifikaciju korisnika. Ove komponente omogućavaju pravilnu obradu zahtjeva i odgovora unutar aplikacije. Kroz *DATABASES* postavku definirani su svi potrebni parametri za spajanje na bazu podataka, uključujući *ENGINE* koji specificira korištenje *PostgreSQL*-a, *NAME* koji označava naziv baze podataka, *USER* i *PASSWORD* koji predstavljaju korisničke vjerodajnice, te *HOST*

i *PORT* koji specificiraju mrežnu adresu poslužitelja baze podataka.Time se osigurava pouzdana i sigurna veza s *PostgreSQL* bazom podataka smještenom na udaljenom poslužitelju.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'expense_tracker_database_hk51',
        'USER': 'expense_tracker_database_hk51_user',
        'PASSWORD': 'gBSsCwy24B5Ruy5YF50DLe9dvRqKbXc9',
        'HOST': 'dpg-coh1tkmv3ddc73ff9vn0-a.frankfurt-postgres.render.com',
        'PORT': '5432'
    }
}
```

Slika 15. Konfiguracija za spajanje na bazu podataka

3.3.2. URLs

Nakon što klijent pošalje HTTP zahtjev prema poslužitelju, prvo se provodi obrada u datoteci *urls.py.* Django koristi ovu datoteku za određivanje kojem će pogledu zahtjev biti proslijeđen. Sustav usmjeravanja URL-ova (URL routing) u urls.py uspoređuje putanju zatraženu u zahtjevu s definiranim uzorcima putanja i traži odgovarajuću podudarnost. Kada se pronađe podudaranje, Django prosljeđuje zahtjev odgovarajućem pogledu zajedno s dodatnim parametrima izvađenim iz URL-a, ako je to potrebno.

U našem projektu imamo dvije takve datoteke, jedna u *users* aplikaciji, a jedna u *api* aplikaciji. Na slikama 16. i 17 . nalazi se kod iz spomenutih datoteka.

```
1  from django.urls import path
2  from rest_framework_simplejwt.views import TokenRefreshView
3
4  from users import views
5
6  app_name = "users"
7
8  urlpatterns = [
9      path("register/", views.UserRegisterationAPIView.as_view(), name="create-user"),
10     path("login/", views.UserLoginAPIView.as_view(), name="login-user"),
11     path("token/refresh/", TokenRefreshView.as_view(), name="token-refresh"),
12     path("logout/", views.UserLogoutAPIView.as_view(), name="logout-user"),
13     path("", views.UserAPIView.as_view(), name="user-info"),
14     path("profile/", views.UserProfileAPIView.as_view(), name="user-profile"),
15 ]
```

Slika 16. Datoteka urls.py iz "users" aplikacije

```
router = DefaultRouter()
router.register(r'categories', CategoryViewSet, basename='category')
router.register(r'accounts', AccountViewSet, basename='account')
router.register(r'transaction-types', TransactionTypeViewSet, basename='transactiontype')
router.register(r'transactions', TransactionViewSet, basename='transaction')
router.register(r'recurring_transactions', RecurringTransactionViewSet)
router.register(r'dailyaccountbalance', DailyAccountBalanceViewSet, basename='dailyaccountbalance')

urlpatterns = [
    path('', include(router.urls)),
]
```

Slika 17. Datoteka urls.py iz "api" aplikacije

3.3.3. Modeli

U poslužiteljskom dijelu aplikacije, ključna komponenta je domenski model koji omogućuje preslikavanje objekata iz programskog koda napisanog u Pythonu u relacijsku bazu podataka. Kako bi se to postiglo, stvaraju se domenske klase koje nasljeđuju *models.Model* klasu, što označava da predstavljaju entitete u bazi podataka. Svaka domenska klasa koja nasljeđuje *models.Model* mora imati primarni ključ, koji se automatski generira kao polje *id* od strane Djanga. Za generiranje vrijednosti primarnog ključa, koristi se polje *AutoField* koje automatski generira vrijednost za taj atribut.

Django koristi vlastiti ORM (objektno relacijsko mapiranje) koje omogućuje rad s bazom podataka koristeći Python objekte i metode umjesto pisanja direktnih SQL upita. ORM mapira Python klase na relacijske tablice i automatski generira SQL upite na temelju definiranih modela. Ovo nam omogućuje da pišemo čist i održiv kod, dok se Django brine o detaljima komunikacije s bazom podataka. ORM također omogućuje definiranje složenih odnosa između modela, kao što su jedan-prema-jedan, jedan-prema-više i više-prema-više. Modeli koji su definirani u našoj aplikaciji su: Category, Account, TransactionType, Transaction, RecurringTransaction, DailyAccountBalance i User.

```python
class Transaction(models.Model):
    id = models.AutoField(primary_key=True)
    amount = models.IntegerField()
    description = models.CharField(max_length=200, null=True, blank=True)
    execution_date = models.DateTimeField(default=timezone.now)
    recurrence_frequency = models.IntegerField(null=True, blank=True)
    transaction_type = models.ForeignKey(TransactionType, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    account = models.ForeignKey(Account, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.description} - {self.amount}"
```

Slika 18. Primjer modela "Transaction"

Model *Transaction* u Django aplikaciji predstavlja entitet transakcije u bazi podataka. Svaka instanca ove klase predstavlja pojedinačnu transakciju. Evo objašnjenja svakog atributa ovog modela:

- *id*: Ovo je automatski generirano polje koje služi kao primarni ključ. Koristi *AutoField*, što znači da Django automatski generira jedinstvene vrijednosti za svaku instancu modela.
- *amount*: Ovo polje predstavlja iznos transakcije. Koristi *IntegerField* za pohranu cijelih brojeva.
- *description*: Ovo polje sadrži opis transakcije. *CharField* je ograničeno na maksimalno 200 znakova, a može biti *null* ili *blank* (nije obavezno).
- *execution_date*: Ovo polje sadrži datum i vrijeme izvršenja transakcije. *DateTimeField* je inicijalizirano na trenutno vrijeme prilikom stvaranja nove transakcije pomoću *default=timezone.now*.
- *recurrence_frequency*: Ovo polje označava učestalost ponavljanja transakcije (ako je primjenjivo). *IntegerField* može biti *null* ili *blank* (nije obavezno).
- *transaction_type*: Ovo polje je vanjski ključ (*ForeignKey*) koji povezuje transakciju s tipom transakcije.
- *user*: Ovo polje je vanjski ključ koji povezuje transakciju s korisnikom koji ju je izvršio. *settings.AUTH_USER_MODEL* referencira korisnički model aplikacije.
- category: Ovo polje je vanjski ključ koji povezuje transakciju s kategorijom kojoj pripada.

- account: Ovo polje je vanjski ključ koji povezuje transakciju s računom s kojeg je izvršena.

Kod atributa *transaction_type*, *user*, *category* i *account*, koristimo *on_delete=models.CASCADE*. To znači da će brisanje bilo kojeg objekta na koji ovi atributi ukazuju (npr. tip transakcije, korisnik, kategorija, račun) automatski izbrisati sve povezane transakcije.

3.3.4. Serijalizatori

Datoteka serializers.py omogućava serijalizaciju i deserijalizaciju podataka. Serijalizacija je proces pretvaranja složenih tipova podataka, poput instanci Django modela, u jednostavnije tipove, obično u formatu JSON, kako bi se podaci mogli lako prenositi preko HTTP protokola. Deserijalizacija je obrnuti proces pri kojem se JSON podaci primljeni putem HTTP zahtjeva pretvaraju natrag u složene Python tipove koje Django može koristiti. Ovo uključuje validaciju primljenih podataka i njihovo pretvaranje u modele prije nego što se spreme u bazu podataka.

```
from rest_framework import serializers
class TransactionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Transaction
        fields = ['id', 'amount', 'description', 'execution_date',
                  'recurrence_frequency', 'transaction_type', 'user', 'category', 'account']
        read_only_fields = ['user',]
```

Slika 19. Primjer serijalizera

Na slici 19. nalazi se primjer jednog serijalizera. *TransactionSerializer* nasljeđuje *ModelSerializer* i automatski preslikava *Transaction* model u JSON format. Klasa *Meta* specificira model i polja koja će biti uključena, dok *read_only_fields* određuje da je polje *user* samo za čitanje, sprječavajući korisnike da mijenjaju ovo polje putem API-ja.

3.3.4. Pogledi

U Django aplikaciji, pogled (*view*) je funkcija ili metoda koja prima HTTP zahtjev i vraća HTTP odgovor. Pogledi su odgovorni za obradu zahtjeva, dohvaćanje podataka iz modela, validaciju podataka, obradu poslovne logike i vraćanje odgovarajućeg odgovora korisniku. Pogledi se mogu definirati kao funkcijski pogledi (*function-based views*) ili pogledi klase (*class-based views*).

ViewSet je posebna vrsta pogleda u Django REST Framework-u koja pruža način za grupiranje srodnih pogleda u jedinstveni set. *ViewSet*-ovi omogućuju definiranje logike za različite HTTP

metode (GET, POST, PUT, DELETE) unutar jedne klase, pojednostavljujući tako rad s RESTful API-jem. Nasljeđuju ViewSet klasu iz rest_framework.viewsets.

ModelViewSet je proširenje *ViewSet*-a koje dodatno pojednostavljuje rad s Django modelima. Automatski pruža osnovne CRUD (*Create*, *Retrieve*, *Update*, *Delete*) operacije za zadani model bez potrebe za pisanjem dodatnog koda. Nasljeđuje *ModelViewSet* klasu iz *rest_framework.viewsets*.

Evo detaljnijeg opisa svake metode:

- **.list()** - Ova metoda se koristi za dohvat liste objekata. Odgovara *HTTP GET* zahtjevu na URL-u koji predstavlja kolekciju resursa. Metoda koristi *serializer* za formatiranje podataka koji se vraćaju korisniku.
- **.retrieve()** - Metoda se koristi za dohvat specifičnog objekta po njegovom identifikacijskom ključu. Odgovara *HTTP GET* zahtjevu na URL-u koji uključuje ID specifičnog objekta.
- **.create()** - Ova metoda omogućava kreiranje novog objekta. Odgovara HTTP POST zahtjevu i očekuje podatke potrebne za kreiranje novog objekta u zahtjevu. Koristi *serializer* za validaciju i spremanje novog objekta u bazu podataka.
- **.update()** - Koristi se za ažuriranje postojećeg objekta. Odgovara HTTP PUT zahtjevu i očekuje potpune nove podatke za objekt koji se ažurira. Metoda također koristi *serializer* za validaciju podataka i ažuriranje objekta u bazi.
- **.partial_update()** - Slično kao .update(), ali koristi HTTP PATCH zahtjev koji omogućuje parcijalno ažuriranje objekta. To znači da se mogu poslati i promijeniti samo određeni dijelovi podataka, umjesto da se moraju slati kompletni novi podaci.
- **.destroy()** - Ova metoda omogućava brisanje objekta. Odgovara HTTP DELETE zahtjevu i zahtijeva identifikacijski ključ objekta koji se briše. Nakon što korisnik pošalje zahtjev za brisanje, objekt se uklanja iz baze podataka.

Slika 20. prikazuje primjer jedne metode koja je dio TransactionViewSet-a.

Metoda prvo dohvaća objekt *user* iz zahtjeva kako bi identificirala korisnika koji je poslao zahtjev. Početni *queryset* se stvara korištenjem *Transaction.objects.filter(user=user),* čime se osigurava da rezultati uključuju samo transakcije koje pripadaju tom korisniku.

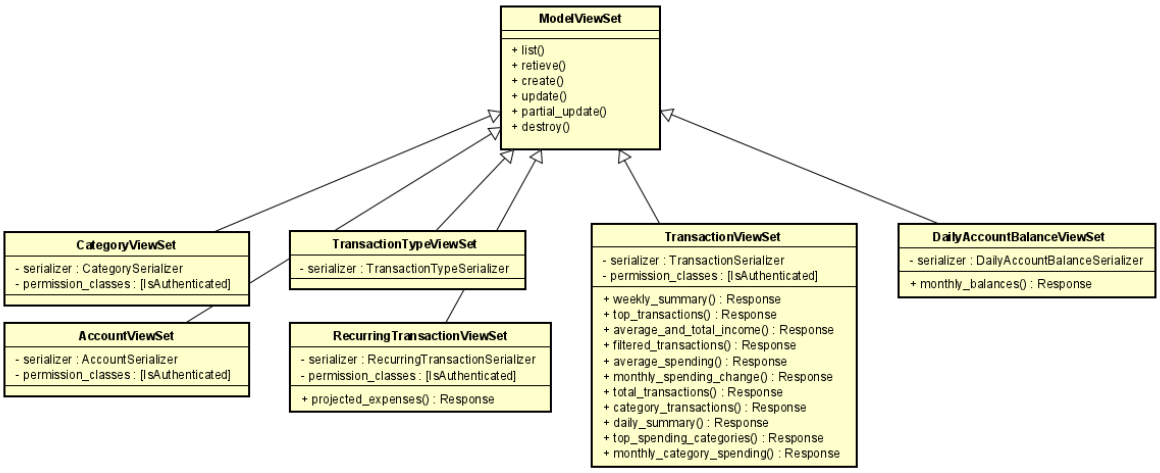
Metoda provjerava prisutnost parametara *start_date* i *end_date* u *request.query_params*. Ako su oba datuma navedena, koristi se *parse_datetime* za konverziju *stringova* datuma u objekte

datetime, a zatim se koristi *make_aware* za osiguravanje svjesnosti o vremenskoj zoni. *Queryset* se tada dodatno filtrira kako bi uključio samo transakcije izvršene unutar definiranog raspona datuma. Dodatno, ako je specificiran *category_id* u parametrima zahtjeva, *queryset* se još filtrira za uključivanje transakcija koje pripadaju određenoj kategoriji.

```python
class TransactionViewSet(viewsets.ModelViewSet):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        """
        This method returns a list of transactions for the currently authenticated user
        that are optionally filtered by an execution date range and a specific category.
        """
        user = self.request.user
        queryset = Transaction.objects.filter(user=user)
        start_date = self.request.query_params.get('start_date')
        end_date = self.request.query_params.get('end_date')
        category_id = self.request.query_params.get('category_id')

        if start_date and end_date:
            start_date = make_aware(parse_datetime(start_date))
            end_date = make_aware(parse_datetime(end_date))
            queryset = queryset.filter(execution_date__range=(start_date, end_date))

        if category_id:
            queryset = queryset.filter(category_id=category_id)

        return queryset
```

Slika 20. Primjer metode za dohvaćanje podataka

Na slici 21. nalazi se dijagram koji prikazuje sve poglede.



Slika 21. Dijagram pogleda

U nastavku slijedi kratak opis svake od metoda koje se nalaze u prethodno prikazanom dijagramu. Metode koje su naslijeđene od *ModelViewSet* klase neće biti dodatno opisivane je su njihove funkcionalnosti već opisane te se ne mijenjaju za svaki *viewset* posebno.

CategoryViewSet

- **get_queryset()**
  Metoda vraća listu kategorija za trenutno autentificiranog korisnika. Kategorije se filtriraju pomoću identifikacijskog ključa korisnika do kojeg se dolazi kroz objekt *user* koji se nalazi u poslanom zahtjevu

AccountViewSet

- **get_queryset**
  Metoda vraća listu računa za trenutno autentificiranog korisnika. Računi se filtriraju pomoću identifikacijskog ključa korisnika do kojeg se dolazi kroz objekt *user* koji se nalazi u poslanom zahtjevu

ReccuringTransactionViewSet

- **daily_summary()**
  Metoda prvo identificira korisnika iz zahtjeva i koristi ga za filtriranje transakcija koje su specifične za tog korisnika. Zatim provjerava jesu li u zahtjevu definirani početni i završni datum te ih u tom slučaju obrađuje, pretvarajući ih u objekte *datetime*, omogućavajući filtriranje transakcija unutar određenog raspona datuma. Provjerava prisutnost parametra *category_id* te u slučaju njegova postojanja filtrira podatke samo za zadane kategorije. Metoda također nudi mogućnost filtriranja podataka na temelju vrste transakcije: prihod ili trošak. Cijeli postupak rezultiramo listom objekata koji predstavljaju jedan dan koji se sastoji od liste izvršenih transakcija te zbroj troškova i prihoda za taj dan.

- **projected_expenses()**
  Metoda *projected_expenses* prvo identificira korisnika iz zahtjeva i koristi ga za filtriranje ponavljajućih transakcija koje su specifične za tog korisnika i čiji se sljedeći datum izvršenja nalazi prije kraja sljedećeg mjeseca, a koje nemaju definiran datum završetka ili je taj datum izvan sljedećeg mjeseca. Zatim metoda obrađuje svaku transakciju na temelju njene frekvencije izvršenja kako bi izračunala koliko puta će se

transakcija izvršiti unutar sljedećeg mjeseca. Metoda sumira sve iznose iz tih transakcija kako bi pružila ukupnu projiciranu vrijednost troškova za sljedeći mjesec.

DailyBalanceViewSet

- **monthly_balances()**

  Metoda monthly_balances provjerava jesu li dostupni i ispravni potrebni parametri, kao što su ID računa i godina. U slučaju da je u zahtjevu specificiran i mjesec, metoda dohvaća dnevna stanja računa za zadani mjesec. metoda dohvaća stanje za posljednji dan svakog mjeseca tijekom cijele godine. U konačnici metoda vraća listu dnevnih stanja odabranog računa za odabrano vremensko razdoblje.

TransactionViewSet

- **get_querySet()**

  Metoda get_queryset vraća popis transakcija koje su povezane s trenutno autentificiranim korisnikom. Osnovni upit filtrira transakcije prema korisniku. Dodatno, metoda omogućuje filtriranje transakcija prema rasponu datuma izvršenja i specifičnoj kategoriji, ako su ti parametri navedeni.

- monthly_category_spending()

  Metoda *monthly_category_spending* vraća podatke o troškovima po kategorijama za trenutni i prethodni mjesec, formatirane za usporedbu pomoću grafikona. Najprije dohvaća sve kategorije koje pripadaju trenutno autentificiranom korisniku. Zatim izračunava ukupne troškove za svaku kategoriju u tekućem mjesecu te idućem mjesecu. Podaci uključuju ukupne iznose potrošnje po kategorijama za oba mjeseca, pri čemu se za svaku kategoriju dodaje i reprezentativna boja.

- weekly_summary()

  Metoda *weekly_summary* pruža pregled troškova za tekući tjedan, grupiranih po danima. Dohvaćaju se sve transakcije označene kao 'troškovi' za trenutno autentificiranog korisnika koje su se dogodile unutar trenutnog tjedna. Transakcije se grupiraju po danu izvršenja, a zatim se sumiraju troškovi za svaki dan. Rezultat je lista koja za svaki dan u tjednu prikazuje iznos ukupnih troškova.

- category_expenses()

Metoda *category_expenses* omogućava analizu troškova po kategorijama za određeni vremenski period koji može biti dan, tjedan ili mjesec. Nakon što korisnik odabere željeni vremenski period, metoda postavlja odgovarajuće početne i završne datume. Zatim filtrira transakcije koje pripadaju autentificiranom korisniku, ograničene na navedeni vremenski raspon i označene kao troškovi. Troškovi se agregiraju po kategorijama, a za svaku kategoriju izračunava se ukupan iznos troškova te njihov udio u ukupnom zbroju troškova u navedenom periodu.

- top_spending_categories()

  Metoda *top_spending_categories* počinje određivanjem vremenskog raspona za prethodni mjesec, zatim se filtriraju transakcije koje su označene kao troškovi. Troškovi se agregiraju po kategorijama, a rezultati se sortiraju prema ukupnoj potrošnji od najveće do najmanje. Metoda završava dohvaćanjem podataka za tri kategorije s najvećom potrošnjom, koje se prikazuju korisniku, uključujući ID kategorije, naziv, boju i ukupan iznos potrošnje za svaku od top tri kategorije.

- daily_summary()

  Metoda ima jednaku funkcionalnost kao istoimena metoda u *ReccuringTransactionViewSetu,* samo što ova vraća podatke vezane za obične transakcije.

- category_transactions()

  Metoda *category_transactions* vraća popis transakcija filtriran po odabranoj kategoriji i vremenskom razdoblju. Transakcije se sortiraju prema datumu izvršenja od najnovijeg prema najstarijem.

- monthly_spending_change()

  Metoda *monthly_spending_change* agregira ukupne troškove po kategorijama za trenutni i prethodni mjesec koristeći transakcije koje su označene kao troškovi. Za svaku kategoriju, metoda izračunava postotnu promjenu potrošnje uspoređujući troškove posljednjeg mjeseca s mjesecom prije njega. Ako u prethodnom mjesecu nema troškova, postotna promjena se postavlja na nulu.

- average_spending()

  Metoda *average_spending* započinje dohvaćanjem ukupnih troškova koji su se izvršili u proteklih godinu dana. Zatim izračunava broj tjedana i mjeseci unutar tog vremenskog razdoblja kako bi omogućila precizan proračun prosječnih vrijednosti, odnosno, u obzir neće uzimati tjedne ili mjesece u kojima nema evidentiranih troškova.

- average_and_total_income()

  Metoda *average_and_total_income* dohvaća sve transakcije kategorizirane kao prihodi unutar definiranog razdoblja i izračunava ukupne prihode. Na temelju broja proteklih mjeseci u tekućoj godini, metoda proračunava prosječni mjesečni prihod.

- top_transactions()

  Metoda *top_transacions* pruža pregled deset najvećih rashodnih transakcija za određeni račun i vremenski period. Počinje provjerom obaveznih parametara: ID računa, godine, i opcionalno mjeseca. U obzir uzima transakcije označene kao troškovi na zadanom računu izvršenu odabranom vremenskom rasponu, te se rezultati sortiraju po iznosu.

3.3.5. Periodični zadaci

Određeni aspekti aplikacije zahtijevaju definiranje zadataka koji će se izvršavati periodično kako bi se osiguralo redovito ažuriranje podataka i optimizacija upita. Za implementaciju ovih zadataka koristi se biblioteka *APScheduler*.

APScheduler (*Advanced Python Scheduler*) je Python biblioteka koja omogućava definiranje zadataka. U konkretnom slučaju, koristi se *BackgroundScheduler*, koji je koristan za planiranje periodičnih zadataka bez ometanja glavnog procesa servera, što ga čini idealnim za zadatke kao što su dnevna ažuriranja ili redovite provjere. *CronTrigger* se koristi za definiranje vremena izvršavanja zadatka koristeći sintaksu sličnu *cronu*, što omogućuje precizno planiranje zadataka.

U aplikaciji su definirana dva takva zadatka: *record_daily_balances* i *execute_recurring_transactions*.

- **record_daily_balances**

  Zadatak record_daily_balances automatski bilježi dnevne stanja računa u bazu podataka svakog dana za sve korisničke račune. Razlog definiranja ovog zadatka je da bi spriječila potreba za retroaktivnim računanjem stanja računa jer se ti podaci koriste za prikaz kretanja stanja pojedinog računa u određenom vremenskom razdoblju. Na ovaj način se smanje količina potrebnog računanja te se bitno smanjuje vrijeme izvođenja spomenutog upita.

- **execute_recurring_transactions**

  Zadatak *execute_recurring_transactions* automatizira proces izvršavanja ponavljajućih transakcija na dan njihova predviđenog izvršenja. a početku, funkcija identificira današnji datum i zatim dohvaća sve instance ponavljajućih transakcija (*RecurringTransaction*) koje treba izvršiti tog dana. Za svaku pronađenu transakciju, funkcija stvara novu transakciju (*Transaction*) i stvara novi zapis u bazi podataka. Nakon stvaranja novog zapisa, ažurira se atribut *nextExecutionDate* ponavljajuće transakcije.

  Na slici 22. nalazi se isječak koda u kojem je prikazana konfiguracija opisanih periodičnih zadataka. Zadatak *execute_recurring_transactions* pokreće se jednom dnevno, u 00:01. Zadatak record_daily_balances također se izvršava jednom dnevno, u 23:55. Svaki zadatak ima definiran jedinstveni identifikator i zadan maksimalni broj istovremenih instanci.

```python
class ApiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'api'

    def ready(self):
        if settings.DEBUG:
            from .tasks import execute_recurring_transactions, record_daily_balances

            scheduler = BackgroundScheduler()
            scheduler.add_job(
                execute_recurring_transactions,
                trigger=CronTrigger(hour=0, minute=1),
                id='recurring_transactions_job',
                max_instances=1,
                replace_existing=True
            )
            scheduler.add_job(
                record_daily_balances,
                trigger=CronTrigger(hour=23, minute=55),
                id='daily_balance_job',
                max_instances=1,
                replace_existing=True
            )

            scheduler.start()
```

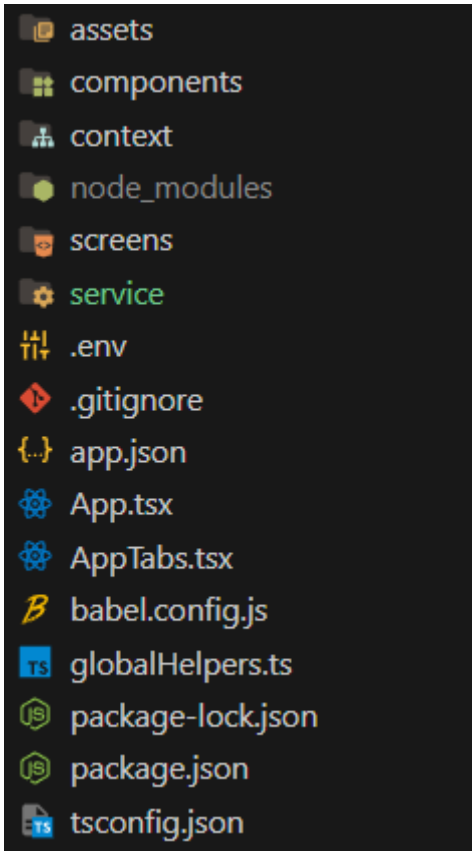
Slika 22. Definiranje periodičnih zadataka

3.4. Arhitektura klijenta

Klijent za naš sustav implementiran je korištenjem React Nativea. React Native je JavaScript okvir za izradu mobilnih aplikacija koji omogućuje kreiranje *nativnih* aplikacija za iOS i Android platforme. React Native koristi virtualni DOM za učinkovito ažuriranje korisničkog sučelja. Aplikacije napisane u React Nativeu koriste komponente za izradu korisničkih sučelja koje se mogu koristiti više pita, a sve promjene stanja komponenata automatski se reflektiraju u korisničkom sučelju. U nastavku će kroz nekoliko odlomaka biti dan pregled najvažnijih dijelova klijenta uz objašnjenja načina rada.

3.4.1. Struktura repozitorija

Na slici 23. nalazi se struktura repozitorija klijenta. Različiti dijelovi klijenta koji imaju slične funkcionalnosti grupirani su zajedno čime se postiže čime se postiže bolja organizacija koda i lakše održavanje aplikacije. Grupiranjem sličnih komponenti smanjuje se redundancija koda i povećava ponovno korištenje koda, što rezultira učinkovitijim razvojnim procesom.



Slika 23. Struktura repozitorija klijenta

- **asssets**

  Repozitorij „assets“ sadrži statičke resurse aplikacije. Statički resursi jesu slike, ikone i fontovi. U ovom repozitoriju se također nalazi datoteka „colors.ts“ u kojoj su definirani heksadecimalni kodovi koji predstavljaju boje koje se koriste u aplikaciji što omogućava da se na jednom mjestu promijeni čitava shema boja.

- **components**

  Repozitorij „components“ sadrži sve komponente koje se koriste unutar aplikacije. Konkretno, sadrži repozitorij „modals“ u kojem se nalaze komponente koje omogućavaju prikaz skočni prozora odnosno dijaloga u aplikaciji koji služe za unos novih transakcija, novih kategorija ili potvrdu određene akcije poput brisanja. Sadrži i komponente koje se koriste u statističkom dijelu aplikacije, a to su komponente za prikaz različitih dijagrama.
- **context**

  Repozitorij „context“ sadrži kontekstne module koji omogućuju dijeljenje globalnog stanja i funkcionalnosti između različitih dijelova aplikacije koristeći Zustand biblioteku. To omogućava učinkovito upravljanje stanjem aplikacije i prijenos podataka bez potrebe za kaskadnim propagiranjem tih podataka kroz komponente.

- screens

  Repozitorij „screens“ sadrži sve zaslone aplikacije. Zasloni su komponente koje predstavljaju različite prikaze unutar aplikacije, poput početne stranice, stranice s transakcijama i slično. Svaki zaslon obično sadrži više komponenti i predstavlja logički odvojenu cjelinu unutar aplikacije.

- service

  Repozitorij „service“ sadrži poslovnu logiku odnosno API pozive koji komuniciraju s poslužiteljem. Ovi moduli omogućavaju aplikaciji da obavlja kompleksne operacije i interakcije s podacima. Način na koji se obavlja komunikacija s poslužiteljem bit će opisan u nastavku.

3.4.2. Pokretanje aplikacije

U ovom dijelu ćemo opisati dvije ključne datoteke u React Native aplikaciji koje služe kao početna točka i pokreću cijelu aplikaciju: App.tsx i AppTabs.tsx.

Datoteka **App.tsx** je glavna komponenta aplikacije, koja postavlja osnovnu konfiguraciju za funkcioniranje aplikacije. U njoj se definiraju navigacijski stogovi za prijavu i registraciju (*AuthStack*), te glavne zaslone aplikacije (*AppStack*). Također se postavljaju rukovatelji obavijestima i učitavaju prilagođeni fontovi. Komponenta *AppContent* upravlja stanjem korisnika i dohvaća podatke o korisniku, te definira navigacijsku strukturu ovisno o tome je li korisnik prijavljen.

Isječak koda koji se nalazi na slici 24. omotava glavnu komponentu aplikacije s različitim pružateljima usluga kako bi omogućio pristup globalnim funkcionalnostima. *QueryClientProvider* osigurava da aplikacija koristi *react-query* za upravljanje podacima dohvaćenim iz API-ja, dok *UserProvider* omogućuje dijeljenje informacija o korisniku i upravljanje korisničkim stanjem. *SafeAreaProvider* osigurava da se sadržaj aplikacije pravilno prikazuje unutar sigurnih granica uređaja.

Unutar *SafeAreaView*, koji osigurava da sadržaj zauzima cijeli zaslon, nalazi se glavna komponenta *AppContent*, koja sadrži sadržaj i navigaciju aplikacije.

```
<QueryClientProvider client={queryClient}>
    <UserProvider>
        <SafeAreaProvider>
            <SafeAreaView>
                <AppContent />
            </SafeAreaView>
        </SafeAreaProvider>
    </UserProvider>
</QueryClientProvider>
```

Slika 24. Isječak koda iz App.tsx

Datoteka **AppTabs.tsx** definira navigacijsku strukturu aplikacije putem *tab* navigacije. U njoj se koristi animirane *tab* navigacija s *AnimatedTabBarNavigator*, ikone iz različitih setova (*Feather*, *Ionicons*, *FontAwesome5*, *AntDesign*), te prilagođeni fontovi i boje. Definirana je funkcionalna komponenta *TabBarIcon* koja omogućava prikaz različitih ikona u *tab* navigaciji.

Glavna komponenta *AppTabs* učitava prilagođene fontove i kreira *Tabs.Navigator* koji sadrži definiciju glavnih zaslona aplikacije (*HomeScreen*, *TransactionsScreen*, *StatsScreen*, *ProfileStackScreen*).

3.4.3. Komunikacija s poslužiteljem

Dvije glavne biblioteke koje se koriste za komunikaciju sa poslužiteljem jesu *Axios* i *React Query*.

Axios je standardna biblioteka koja se koristi u većini React Native projekata, riječ je o biblioteci za HTTP zahtjeve u JavaScriptu, koja omogućava slanje asinkronih zahtjeva prema poslužitelju za dohvaćanje ili slanje podataka. Pruža jednostavno sučelje za slanje GET, POST, PUT, DELETE i drugih vrsta zahtjeva, te podržava rad s JSON-om, upravljanje zaglavljima (headers) i rukovanje greškama.

```
import axios from "axios";
import * as SecureStore from "expo-secure-store";

const createAuthenticatedClient = () => {
    const instance = axios.create({
        baseURL: "https://514e-188-129-16-235.ngrok-free.app",
        withCredentials: true,
    });

    instance.interceptors.request.use(
        async (config) => {
            const token = await SecureStore.getItemAsync("accessToken");
            if (token) {
                config.headers = config.headers || {};
                config.headers["Authorization"] = `Bearer ${token}`;
            }
            return config;
        },
        (error) => {
            return Promise.reject(error);
        }
    );

    return instance;
};

export default createAuthenticatedClient;
export const authenticatedClient = createAuthenticatedClient();
export const authclient = axios.create({
    baseURL: "https://514e-188-129-16-235.ngrok-free.app",
    withCredentials: true,
});
```

Slika 25. Konfiguracija axios klijenta

U ovom kodu, funkcija *createAuthenticatedClient* kreira instancu Axios klijenta s unaprijed definiranom *baseURL* adresom i postavkom *withCredentials: true,* što omogućava slanje kolačića zajedno sa zahtjevima.

Instanca Axios klijenta ima definirane *interceptore* za zahtjeve. Prije svakog slanja zahtjeva, *interceptor* dohvaća *accessToken* iz *expo-secure-store* pomoću *SecureStore.getItemAsync("accessToken")*. Ako token postoji, on se dodaje u zaglavlja zahtjeva kao *Authorization* zaglavlje s vrijednošću *Bearer ${token}*. Ovo osigurava da svaki zahtjev nosi odgovarajući token za autentifikaciju, čime se omogućava sigurna komunikacija s poslužiteljem. U slučaju greške prilikom konfiguracije zahtjeva, *interceptor* vraća odbijen *Promise* s greškom.

Na kraju, instanca *Axios* klijenta se vraća i može se koristiti za slanje zahtjeva s autentifikacijom. Pored toga, *authclient* je dodatno definiran za neautentificirane zahtjeve s istom *baseURL* adresom, ali bez automatizirane dodjele *Authorization* zaglavlja.

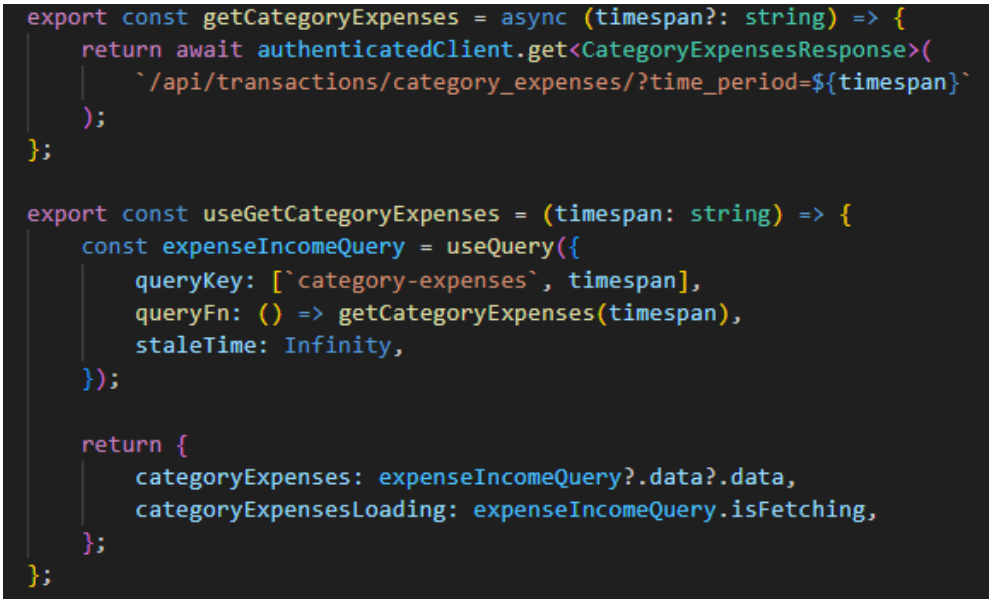
Ovaj način konfiguracije omogućava jednostavno i sigurno upravljanje autentificiranim i neautentificiranim zahtjevima unutar aplikacije, čineći komunikaciju s poslužiteljem efikasnijom i sigurnijom.

React Query je biblioteka čija je glavna svrha pojednostaviti dohvaćanje, predmemoriranje, sinkronizaciju i ažuriranje podataka iz poslužitelja. React Query automatizira dijelove upravljanja podacima i čini rad s asinkronim podacima u React i React Native aplikacijama puno jednostavnijim i učinkovitijim.

React Query omogućava jednostavno dohvaćanje podataka iz API-ja koristeći *useQuery* funkciju. Dohvaćeni podaci se automatski predmemoriraju i mogu se koristiti u različitim dijelovima aplikacije bez potrebe za ponovnim dohvaćanjem. Predmemoriranje podataka smanjuje broj zahtjeva prema poslužitelju i poboljšava performanse aplikacije. React Query automatski upravlja predmemorijom, osiguravajući da su podaci uvijek ažurni i dostupni.

Osim toga, React Query omogućava automatsku sinkronizaciju podataka između klijenta i poslužitelja. Kada se podaci promijene na poslužitelju, React Query će automatski osvježiti predmemorirane podatke na klijentu. Upravljanje stanjem učitavanja i grešaka je također pojednostavljeno. React Query pruža jednostavne metode za prikazivanje stanja učitavanja i rukovanje greškama, što olakšava razvoj i poboljšava korisničko iskustvo.

Podaci se mogu ponovno dohvaćati automatski na različite događaje, poput ponovnog fokusiranja prozora ili ponovnog povezivanja s mrežom. Također je moguće ručno osvježiti podatke prema potrebi.



Slika 26. Primjer slanja GET zahtjeva

Na slici 26. nalazi se primjer slanja GET zahtjeva. Prva funkcija *getCategoryExpenses* koristi instancu Axios klijenta koja je konfigurirana tako da u šalje *accessToken* kako bi poslužitelj znao da je korisnik autentificiran. Poslužitelj prihvaća opcionalni parametar *timespan* koji specificira vremenski period za koji se dohvaćaju podaci o troškovima. Funkcija vraća odgovor od poslužitelja koji sadrži podatke o troškovima po kategorijama.

Druga funkcija *useGetCategoryExpenses* koristi React Query funkciju *useQuery* za dohvaćanje podataka o troškovima po kategorijama. Ova funkcija koristi *getCategoryExpenses* funkciju kao *query* funkciju za dohvaćanje podataka. *useQuery* se konfigurira s *queryKey* koji uključuje znakovni niz *'category-expenses'* i *timespan* kako bi se osiguralo da se podaci predmemoriraju i ponovno dohvaćaju prema tom ključu.

Ovaj pristup omogućava jednostavno dohvaćanje, predmemoriranje i ponovno dohvaćanje podataka o troškovima po kategorijama u React Native aplikaciji, koristeći Axios klijent za HTTP zahtjeve i React Query-a za upravljanje stanjem i predmemoriranjem podataka.

Na slici 27. nalazi se primjer slanja POST zahtjeva.

```
export const useAddTransaction = () => {
    const addTransaction = async (transaction: any) => {
        return await authenticatedClient.post(`/api/transactions/`, {
            amount: transaction.amount,
            description: transaction.description,
            execution_date: transaction.executionDate,
            recurrence_frequency: null,
            transaction_type: transaction.transactionType,
            category: transaction.category,
            account: transaction.account,
        });
    };

    return useMutation(addTransaction, {
        onError: (error) => console.log(error),
    });
};
```

Slika 27. Primjer slanja POST zahtjeva

Funkcija *useAddTransaction* koristi React Query funkciju *useMutation* kako bi upravljala procesom dodavanja nove transakcije. Unutar ove funkcije definirana je asinkrona funkcija *addTransaction*, koja koristi *authenticatedClient* (konfigurirana instanca Axios-a) za slanje POST zahtjeva. Ova funkcija uzima objekt *transaction* koji sadrži detalje transakcije, te se ti podaci se zatim šalju poslužitelju kako bi se stvorila nova transakcija.

React Query funkcija useMutation koristi addTransaction funkciju kao mutacijsku funkciju. *useMutation* također omogućava upravljanje stanjem mutacije i rukovanje greškama. U ovom slučaju, ako dođe do greške tijekom dodavanja transakcije, greška se ispisuje u konzolu putem *onError callback* funkcije.

Na slici 28. nalazi se primjer korištenja mutacije, konkretno za dodavanje nove transakcije.

```
const handleAddTransaction = () => {
    try {
        addTransaction.mutate(
            {
                amount: parseFloat(transactionData.amount),
                description: transactionData.description,
                transactionType: transactionData.transactionType,
                executionDate: transactionData.executionDate,
                frequency: transactionData.frequency,
                account: transactionData.account,
                category: transactionData.category,
            },
            {
                onSuccess: () => {
                    queryClient.invalidateQueries(["daily-summary"]);
                    queryClient.invalidateQueries(["category-expenses"]);
                    queryClient.invalidateQueries(["accounts"]);
                    queryClient.invalidateQueries(["weekly-summary"]);

                    Toast.show({
                        type: "success",
                        text1: "New transaction added!",
                    });
                    handleModalClose();
                },
                onError: () => {
                    Toast.show({
                        type: "error",
                        text1: "Failed to add transaction!",
                    });
                },
            }
        );
    } catch (error) {
        if (error instanceof z.ZodError) {
            const fieldErrors: { [key: string]: string } = {};
            error.errors.forEach((err) => {
                if (err.path) {
                    fieldErrors[err.path[0]] = err.message;
                }
            });
            setErrors(fieldErrors);
        }
    }
};
```

Slika 28. Primjer korištenja mutacije

Funkcija *handleAddTransaction* upravlja procesom dodavanja nove transakcije. Unutar nje, *addTransaction.mutate* se poziva s podacima transakcije, što pokreće mutaciju. Ako je mutacija uspješna, invalidiraju se upiti za osvježavanje podataka (daily-summary, category-expenses, accounts, weekly-summary), prikazuje se uspješna poruka putem Toast.show, i zatvara se modal. Invalidacijom spomenutih upita zapravo se od poslužitelja zahtijevaju novi podaci jer je došlo do stvaranja nove transakcije koja utječe na ispravnost predmemoriranih podataka. Ako dođe do greške, prikazuje se poruka o neuspjehu. U slučaju greške validacije pomoću *Zod* biblioteke, greške se prikupljaju i pohranjuju u *fieldErrors*.

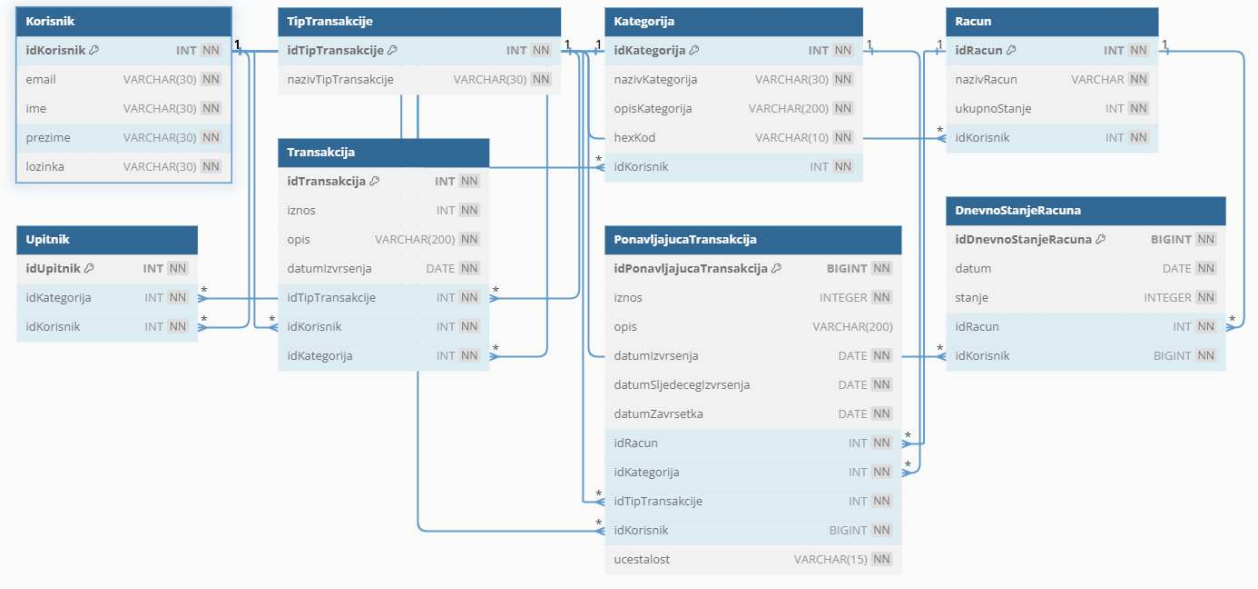
3.4.4. Notifikacije

U ovom radu koriste se biblioteke *expo-notifications* i *expo-constants* za definiranje i upravljanje lokalnim notifikacijama u React Native aplikaciji. *Notifications.setNotificationHandler* postavlja rukovatelja notifikacija koji određuje kako će se notifikacije prikazivati, uključujući tekst i zvuk. Funkcija *schedulePushNotification* zakazuje lokalne notifikacije s različitim frekvencijama (dnevno, svaka dva dana, tjedno i slično) i definira sadržaj notifikacije.

U komponenti *NotificationComponent* koriste se *useState* i *useEffect* za upravljanje stanjem notifikacija i registriranje *listenera* za primanje i odgovaranje na notifikacije. *notificationListener* i *responseListener* slušaju događaje primanja notifikacija i odgovora na njih te se uklanjaju kada komponenta bude postavljena. Također, *registerForPushNotificationsAsync* osigurava da uređaj ima potrebne dozvole za primanje notifikacija. Ovaj pristup omogućava integraciju lokalnih notifikacija u aplikaciji, koristeći funkcionalnosti Expo-a za upravljanje notifikacijama i konfiguraciju uređaja.

3.5. Baza podataka

U ovom dijelu biti će opisana korištena baza podataka. Na slici 29. nalazi se dijagram baze podataka. Dijagram baze podataka je grafički prikaz strukture baze podataka koji prikazuje entitete i odnose među njima. Pomaže u vizualizaciji kako su podaci organizirani i kako su povezani.



Slika 29. Relacijski dijagram baze podataka

Baza podataka sadrži sljedeće entitete, odnosno tablice:

- Tablica Korisnik pohranjuje osnovne podatke o korisnicima, uključujući ID, email, ime, prezime i lozinku. Email je jedinstven za svakog korisnika.
- Tablica TipTransakcije sadrži podatke o vrstama transakcija, uključujući ID i naziv tipa transakcije, gdje je naziv tipa transakcije jedinstven.
- Tablica Kategorija pohranjuje informacije o različitim kategorijama, uključujući ID, naziv, opis, heksadecimalni kod za boje i ID korisnika kojem kategorija pripada.
- Tablica Racun sadrži podatke o računima, uključujući ID, naziv računa, ukupno stanje i ID korisnika koji posjeduje račun.
- Tablica Upitnik povezuje korisnike s kategorijama, sadržeći ID upitnika, ID kategorije i ID korisnika.
- Tablica Transakcija sadrži podatke o transakcijama, uključujući ID transakcije, iznos, opis, datum izvršenja, učestalost izvršavanja, ID tipa transakcije, ID korisnika i ID kategorije.

- Tablica PonavljajucaTransakcija pohranjuje informacije o ponavljajućim transakcijama, uključujući ID, iznos, opis, datum izvršenja, datum sljedećeg izvršenja, datum završetka, ID računa, ID kategorije, ID tipa transakcije, ID korisnika i učestalost.
- Tablica DnevnoStanjeRacuna pohranjuje dnevna stanja računa, uključujući ID, datum, stanje, ID računa i ID korisnika.

Odnosi između entiteta:

- (Korisnik - Kategorija) - jedan korisnik može posjedovati više kategorija, dok jedna kategorija pripada samo jednom korisniku.
- (Korisnik - Racun) - jedan korisnik može posjedovati više računa, dok jedan račun pripada samo jednom korisniku.
- (Korisnik - Upitnik) - jedan korisnik može popuniti više upitnika, dok jedan upitnik pripada samo jednom korisniku.
- (Korisnik - Transakcija) - jedan korisnik može imati više transakcija, dok jedna transakcija pripada samo jednom korisniku.
- (Korisnik - PonavljajucaTransakcija) - jedan korisnik može imati više ponavljajućih transakcija, dok jedna ponavljajuća transakcija pripada samo jednom korisniku.
- (Korisnik - DnevnoStanjeRacuna) - jedan korisnik može imati više dnevnih stanja računa, dok jedno dnevno stanje računa pripada samo jednom korisniku.
- (Kategorija - Upitnik) - jedna kategorija može biti povezana s više upitnika, dok jedan upitnik pripada samo jednoj kategoriji.
- (Kategorija - Transakcija) - jedna kategorija može biti povezana s više transakcija, dok jedna transakcija pripada samo jednoj kategoriji.
- (Kategorija - PonavljajucaTransakcija) - jedna kategorija može biti povezana s više ponavljajućih transakcija, dok jedna ponavljajuća transakcija pripada samo jednoj kategoriji.
- (Racun - PonavljajucaTransakcija) - jedan račun može imati više ponavljajućih transakcija, dok jedna ponavljajuća transakcija pripada samo jednom računu.
- (Racun - DnevnoStanjeRacuna) - jedan račun može imati više dnevnih stanja, dok jedno dnevno stanje pripada samo jednom računu.
- (TipTransakcije - Transakcija) - jedan tip transakcije može biti povezan s više transakcija, dok jedna transakcija pripada samo jednom tipu transakcije.

- (TipTransakcije - PonavljajucaTransakcija) - jedan tip transakcije može biti povezan s više ponavljajućih transakcija, dok jedna ponavljajuća transakcija pripada samo jednom tipu transakcije.

4. Opis korisničkog sučelja

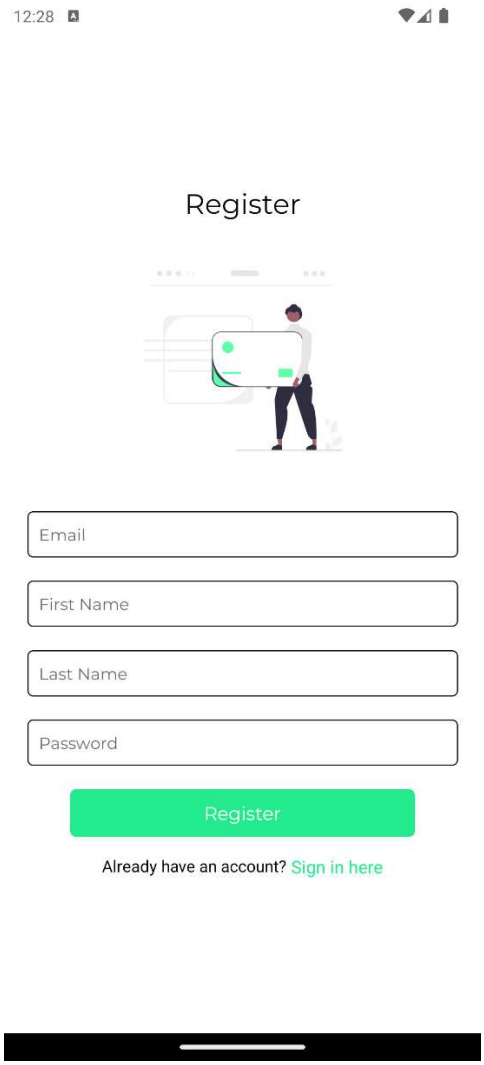
Mobilna aplikacija sastoji se od četiri glavna pogleda odnosno ekrana, od kojih svaki ima svoje specifične funkcionalnosti i način interakcije s korisnikom. U nastavku ovog poglavlja, detaljno će biti opisani svaki od ovih ekrana kako bi se dobio uvid u sve mogućnosti i značajke aplikacije. Osim tekstualnog opisa, uz svaki ekran bit će priložene i slike koje ilustriraju izgled i funkcionalnost aplikacije.

4.1. Pristup aplikaciji – prijava i registracija

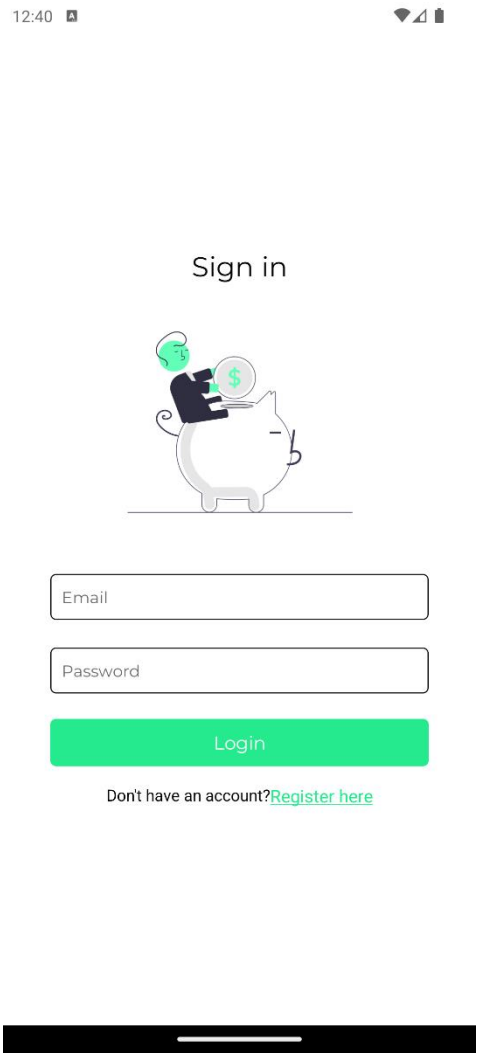
4.1.1. Registracija

Korisnik koji prvi put koristi ovu mobilnu aplikaciju morat će se registrirati kako bi dobio pristup ostalim funkcionalnostima aplikacije. Na slici 30. nalazi se forma za registraciju. Korisnik mora unijeti svoju elektroničku adresu, ime i prezime te lozinku. Lozinka mora imati barem 6 znakova, barem jedno veliko slovo(A-Ž), barem jedno malo slovo(a-ž), barem jednu znamenku(0-9) i barem jedan specijalan znak. Kada su svi podaci ispravno uneseni, korisnik treba dodirnuti dugme „Register“. Ukoliko registracija prođe uspješno, korisnik će biti obaviješten putem obavijesti te će biti preusmjeren na ekran za prijavu u aplikaciju.

U slučaju da korisnik mijenja uređaj, ali je prethodno tome već koristio aplikaciju i registrirani je korisnik, ne mora ponovno raditi registraciju nego dodirom na dugme „Sign in here“ koje se nalazi ispod dugmeta za registraciju može pristupiti prijavi u aplikaciju.



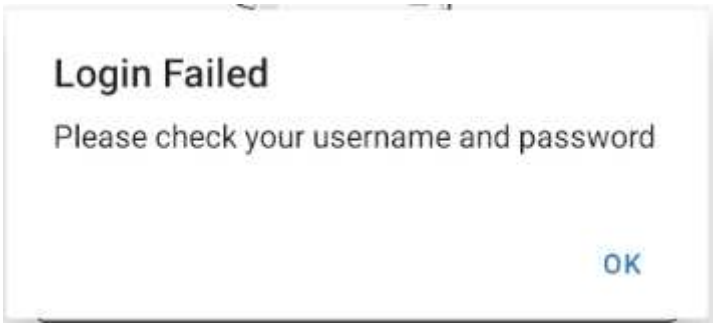
Slika 30. Ekran za registraciju



Slika 31. Ekran za prijavu

4.1.2. Prijava u aplikaciju

Korisnik koji je prethodno izvršio registraciju biti će preusmjeren na ekran za prijavu koji se nalazi na slici 31. Korisnik mora unijeti svoju elektroničku adresu koju je koristio za registraciju te lozinku. Po završetku unosa, potrebno je dodirnuti dugme „Login“. U slučaju da je neka od traženih vjerodajnica pogrešno unesena, korisnik će biti obaviješten putem obavijesti koja je vidljiva na slici 32. Također je bitno za napomenuti da korisnik neće svaki put kada otvori aplikaciju izvršiti prijavu već će se ista dogoditi automatski.



Slika 32. Greška prilikom prijave

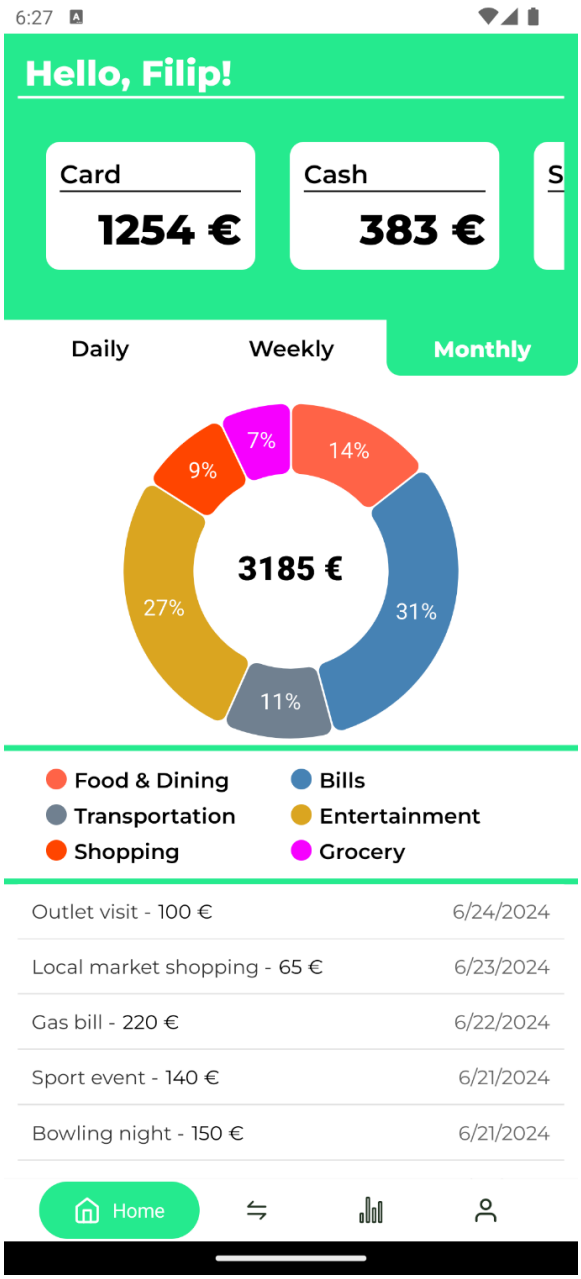
4.2. Početni ekran

Na slici 33. nalazi se početni ekran na koji će korisnik biti preusmjeren nakon uspješne prijave. Na samom vrhu ekrana nalazi se tekst dobrodošlice i korisnikovo ime. Zatim, ispod toga se nalazi segment u kojem su komponente kartica koje predstavljaju račune koje je korisnik dodao u aplikaciju. Svaka kartica na sebi ima naziv računa te trenutno stanje tog računa izraženo u eurima. Moguće je imati više od dva računa, a cijeli segment moguće je dodirom pomaknuti u desno.

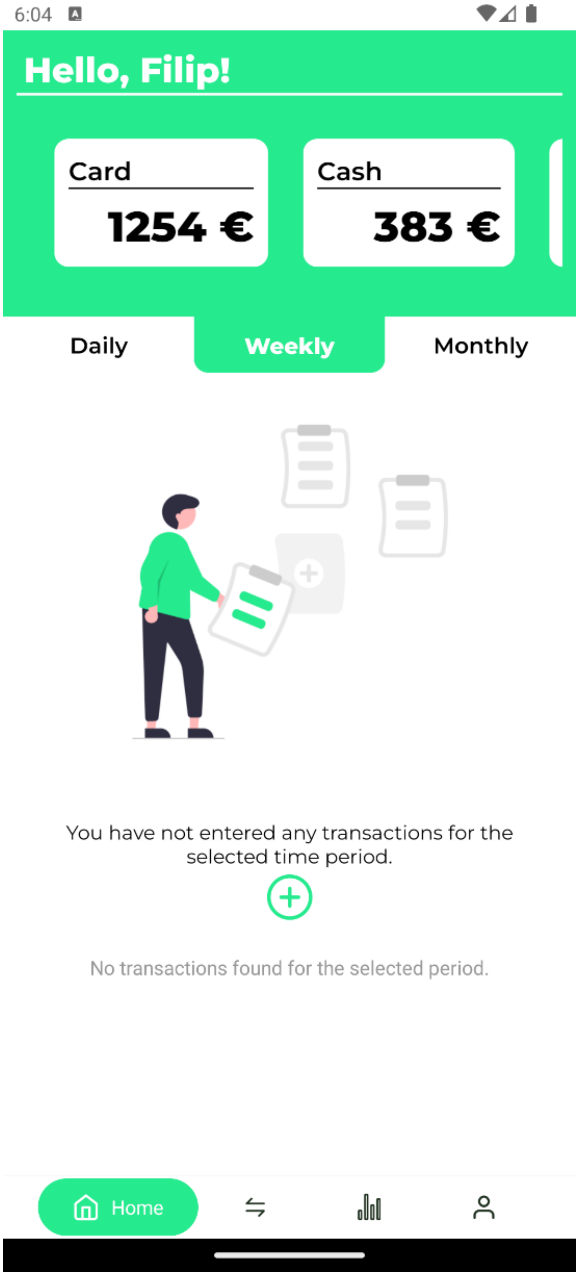
Ispod toga nalazi se dio u kojem je moguće odabrati željeno vremensko razdoblje za koje će se prikazati podaci. Korisnik može birati između tri opcije: dan, tjedan ili mjesec. Ako odabere opciju „dan“ prikazat će mu se podaci za današnji dan, odnosno za tekući tjedan ili mjesec.

Ukoliko korisnik nema unesenih transakcija za odabrano razdoblje, što je trenutno slučaj na slici 34., pokazat će mu se kratka obavijest o tome će se mu se prikazati dugme za unos

transakcije u obliku zelene kružnice sa znakom plus u sredini. Dodirom na to dugme otvara se forma za unos nove transakcije o kojoj će više riječi biti kasnije.



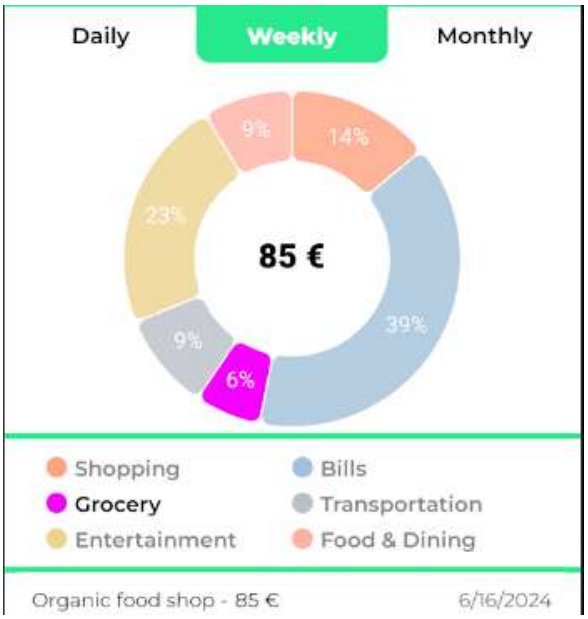
Slika 33. Početni ekran aplikacije



Slika 34. Početni ekran bez transakcija

Nadalje, ispod dijela u kojem je moguće odabrati željeno vremensko razdoblje, nalazi se kružni dijagram u kojem se nalazi kategorizirani prikaz troškova za odabrano vremensko razdoblje. Svaka kategorija koja je zastupljena u transakcijama predstavljena je bojom koja joj je zadana prilikom stvaranja kategorije. Uz boju, na svakom dijelu kružnog dijagrama nalazi se i postotak koji predstavlja udio kategorije u ukupnom zbroju troškova za odabrano razdoblje čiji je iznos prikazan u sredini kružnog dijagrama.

Korisnik ima mogućnost dodirnuti svaku kategoriju u kružnom dijagramu što će rezultirati filtriranjem transakcija koje su prikazane u donjem dijelu ekrana. Ukoliko nije odabrana niti jedna kategorija, tada se prikazuje popis svih transakcija za odabrano razdoblje. Na slici 35. moguće je vidjeti kako izgleda ekran u slučaju da korisnik odabere neku kategoriju.



Slika 35. Kružni dijagram sa odabranom kategorijom

Kada korisnik dodirom odabere neku kategoriju, tada se i iznos koji je prikazan u sredini kružnog dijagrama promijeni te tada prikazuje iznos ukupnog zbroja troškova za odabranu kategoriju.

4.3. Transakcije

Ekran „Transactions“ podijeljen je na dva dijela, „Past“ i „Scheduled“. Prvi prikazuje transkacije koje su se već dogodile, dok drugi dio prikazuje transakcije koje se ponavljaju u određenim vremenskim periodima. Svaki od tih dijelova bit će zasebno opisan.

4.3.1. Izvršene transakcije

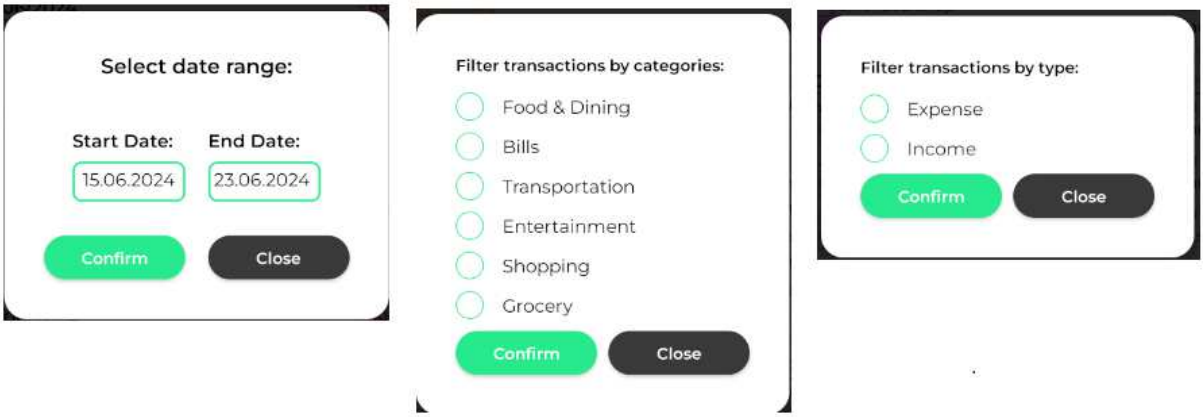
Kao što je već spomenuto, početno stanje ekrana transakcija jest prikaz transakcija koje su se već dogodile. Korisniku se nudi nekoliko alata koji mu omogućavaju lakši pregled popisa transakcija. Prvi od njih je mogućnost pretraživanja. Kao što je vidljivo na slici 36., korisnik može dodirnuti polje za pretraživanje koje se nalazi pri vrhu ekrana u slučaju da traži određenu transakciju.

Nadalje, ispod polja za pretraživanje nalazi se dio u kojem se korisniku nude tri mogućnosti filtriranja prikazanih transakcija. Transakcije je moguće filtrirati po rasponu datuma, kategoriji i tipu transakcije. Na slici 37. vidljivi su dijaloški okviri koji omogućavaju primjenu spomenutih filtera dok se na slici 38. moguće je vidjeti primijenjen filter za kategorije. Svi primijenjeni filteri mogu se poništiti dodirom na crveno dugme „Reset filters“.

Slika 36. Ekran transakcije

Slika 38. Primjena filtera

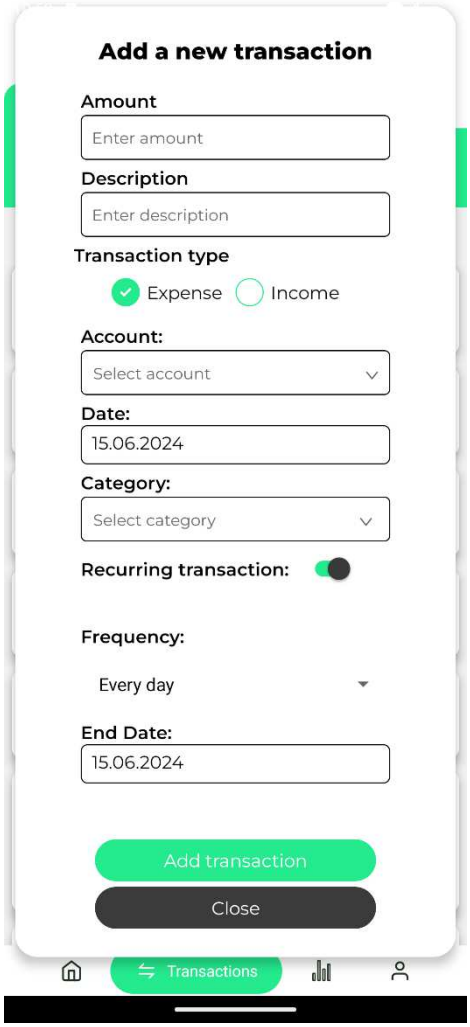
Dodirom na dugme „Add new transaction“ otvara se prozor sa obrascem koji omogućuje dodavanje nove transakcije koji je vidljiv na slici 39. Od korisnika se traži da unese iznos transakcije, opis, tip, račun s kojeg se sredstva uzimaju ili na koji se dodaju, kategorija transakcije. U slučaju da korisnik želi dodati ponavljajuću transakciju, dodirom na dugme „Reccuring transactions“ pojavit će se još dva polja za unos gdje može odabrati učestalost izvođenja te datum završetka koji je opcionalan.



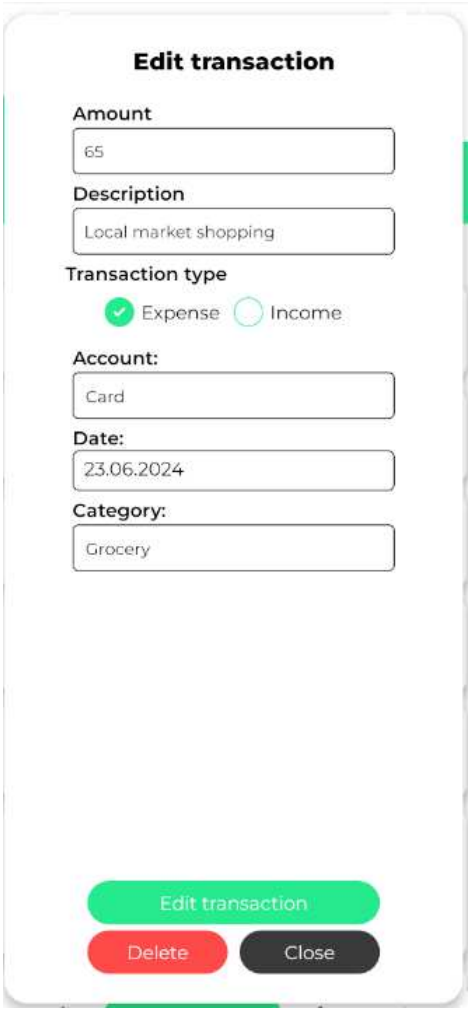
Slika 37. Dijaloški okviri sa opcijama za filtere

Popis transakcija prikazuje niz kartica gdje svaka kartica predstavlja jedan dan, te su u njoj popisane sve transakcije koje su izvedene na taj dan. Na slici 41. vidljiv je primjer jedne takve kartice. Kartica prikazuje datum, zatim popis svih transakcija koje su izvedene s njihovim opisom, kategorijom i iznosom. Ispod iznosa nalazi se naziv računa na kojeg se transakcija odnosi. U gornjem lijevom kutu kartice nalazi se zbroj svih troškova prikazan crvenom bojom i zbroj svih primanja prikazan zelenom bojom.

Dodirom na neku od transakcija koje se nalaze u popisu otvara se prozor za uređivanje već postojeće transakcije. Ovdje je moguće promijeniti neki od parametara transakcije, a moguće je i obrisati odabranu transakciju. Brisanjem transakcije poništava se i njen učinak na stanje računa, što znači da se će se u slučaju da je obrisana transakcija bila trošak, iznos transakcije vratit na račun koji je bio vezan za transakciju, a obrnuti proces vrijedi za transakciju prihoda.



Slika 39. Obrazac za dodavanje transakcije

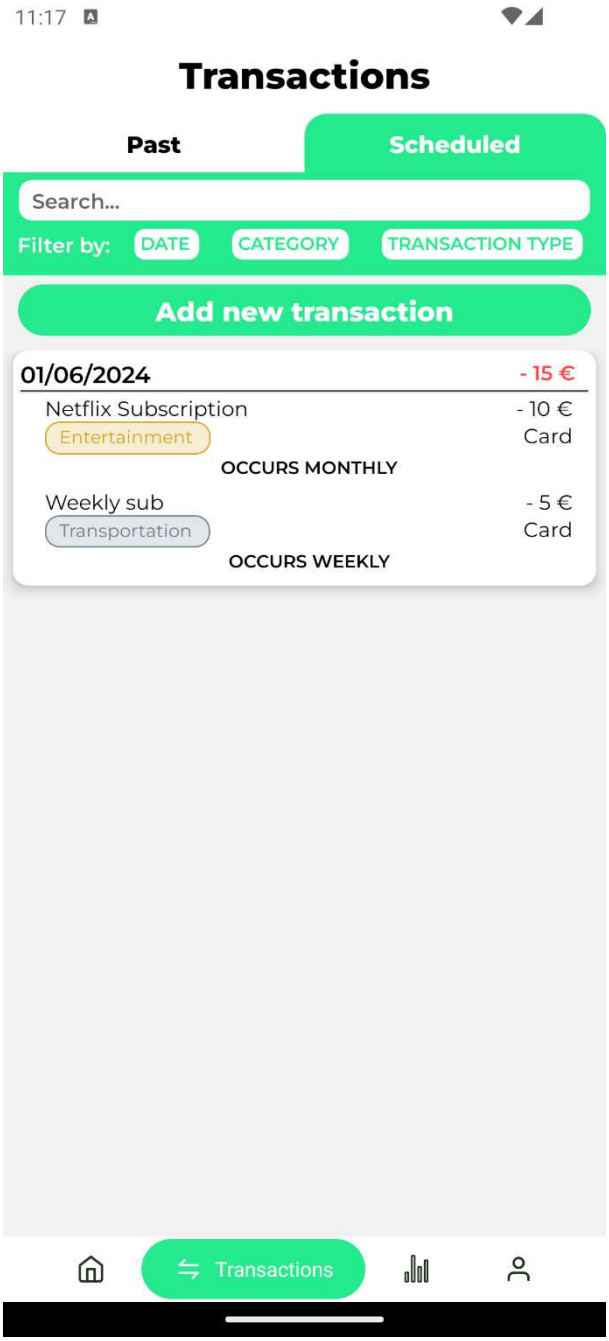


Slika 40. Uređivanje transakcije

Slika 41. Kartica transakcija

4.3.2. Ponavljajuće transakcije

U drugom dijelu ekrana transakcija kojem se pristupa dodirom na jedno od dugmeta o gornjem dijelu ekrana prikazuje ponavljajuće transakcije. Korisniku se u ovom prikazu nude jednake mogućnosti kao i u prethodno opisanom ekranu, a to su mogućnosti pretraživanja, filtriranja po datumu, kategoriji i tipu, mogućnost dodavanja i uređivanja transakcije. U kartici transakcija se dodatno prikazuje učestalost svake od ponavljajućih transakcija.

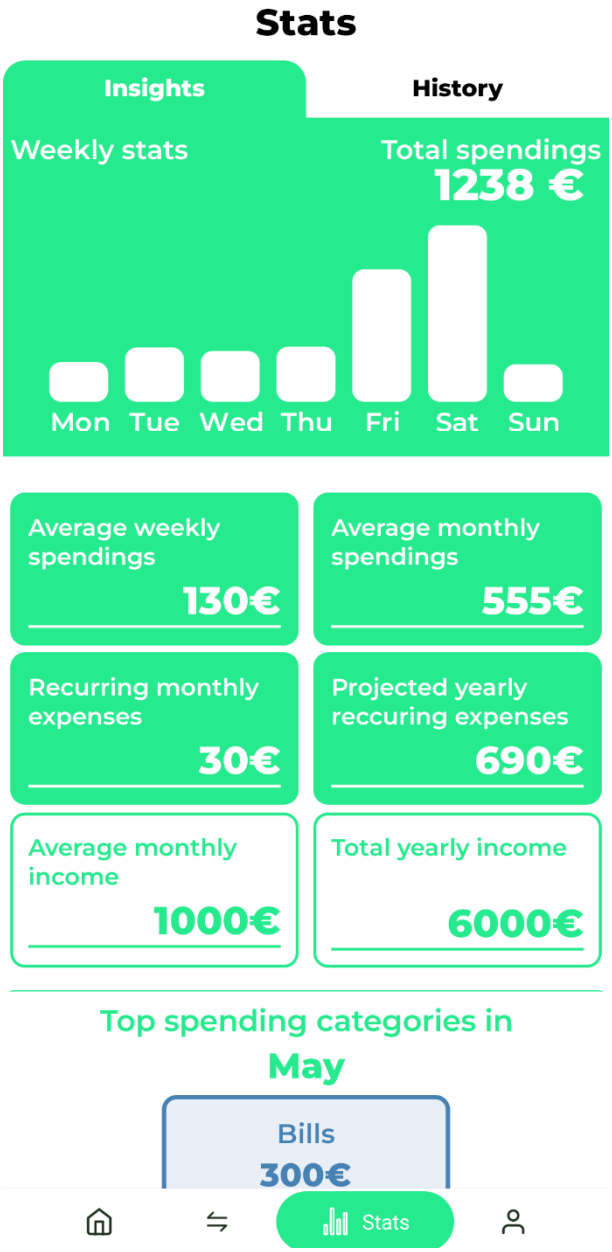


Slika 42. Ponavljajuće transakcije

4.4. Statistika

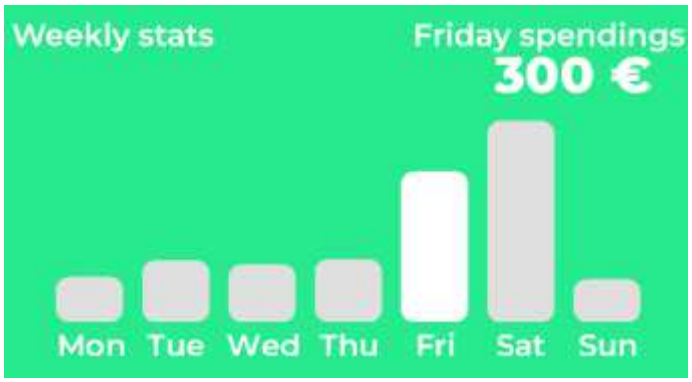
Prikaz statistike podijeljen je na dva dijela, slično kao i transakcije. Riječ je o dijelovima „Insights“ i „History“. Prvi spomenuti prikazuje podatke za trenutni tjedan i prethodni mjesec te razne informacije o prosjeku troškova i primanja. Ideja je da se korisniku ponudi brzi pregled informacije gdje se s njegove strane ne očekuje nikakva akcija. S druge strane, „History“ dio nudi korisniku povijesni pregled stanja određenog računa kroz pregled u dijagramu.

4.4.1. Insights



Slika 43. Statistika

Na slici 43. prikazan je „Insights“ dio ekrana o statistici. U gornjem dijelu nalazi se stupčasti dijagram koji prikazuje troškove za trenutni tjedan. U gornjem lijevom kutu dijagrama početno se nalazi zbroj svih troškova. Korisnik dodirom na jedan od stupaca u dijagramu može doznati iznos troškova za taj odabrani dan kako je vidljivo na slici 44.



Slika 44. Stupčasti dijagram troškova u tekućem tjednu

Nakon opisanog stupčastog dijagrama nalazi se dio u kojem su iznosom prikazani razni statistički podaci kako je vidljivo na slici 45.

Slika 45. Statistički „boxevi“

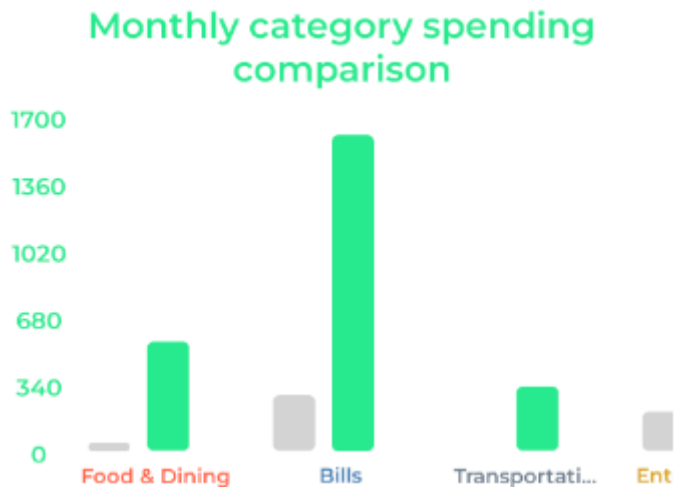
Prikazuju se sljedeći iznosi:

- prosječnih tjednih troškova
- prosjek mjesečnih troškova
- ponavljajućih troškovi za idući mjesec
- predviđeni iznos ponavljajućih troškova za do kraja godine
- prosječni mjesečni prihod
- ukupni godišnji prihod.

Ispod prethodno opisanog prikaza nalazi se kompaktan prikaz tri kategorije za koje je korisnik u prethodnom mjesecu registrirao najviše troškova. Na slici 46. vidljiv je taj prikaz, za svaku kategoriju prikazan je iznos troškova, naziv kategorije te je vizualno predstavljena odabranom bojom.

Slika 46. Kategorije s najvećim troškovima u prethodnom mjesecu

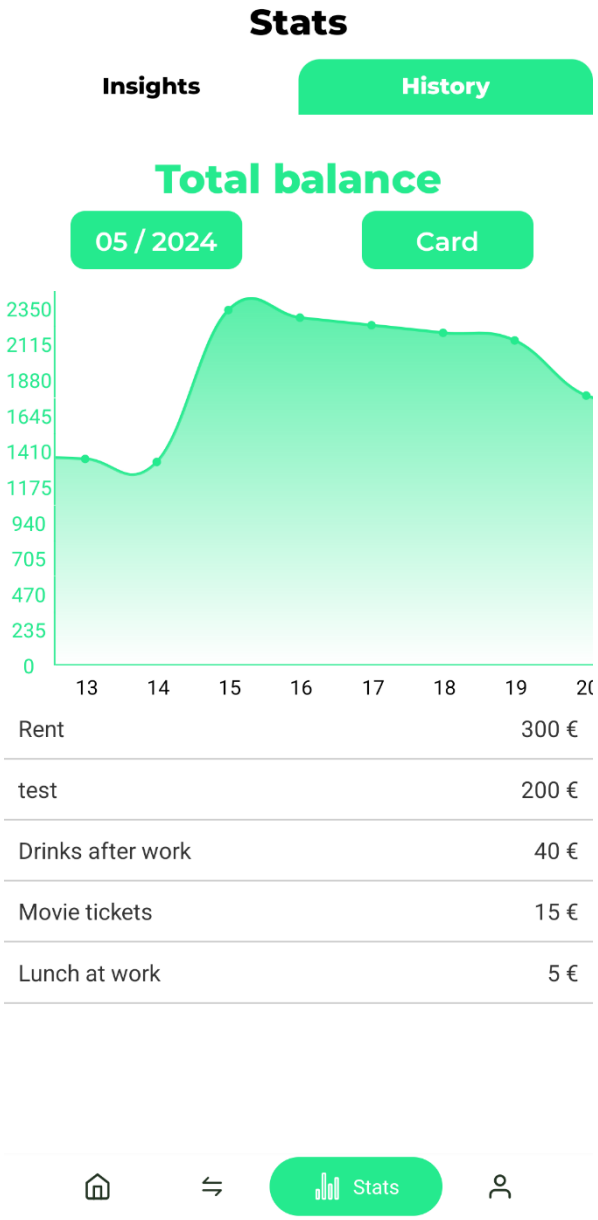
Zadnji dio „Insights“ prikaza jest stupčasti dijagram koji prikazuje usporedbu potrošnje po kategorijama za tekući i prethodni mjesec. Na x osi dijagrama nalaze se kategorije, a na y osi nalaze se iznosi u eurima. Svaka kategorija predstavljena je s dva stupca od kojih jedan prikazuje troškove u toj kategoriji za prethodni mjesec, a jedan za tekući mjesec. Svaki od stupaca moguće je dodirnuti te će se tada prikazati iznos troškova u toj kategoriji iznad stupca. Na slici 47. nalazi se opisani prikaz.



Slika 47. Stupčasti dijagram usporedbe mjesečne potrošnje

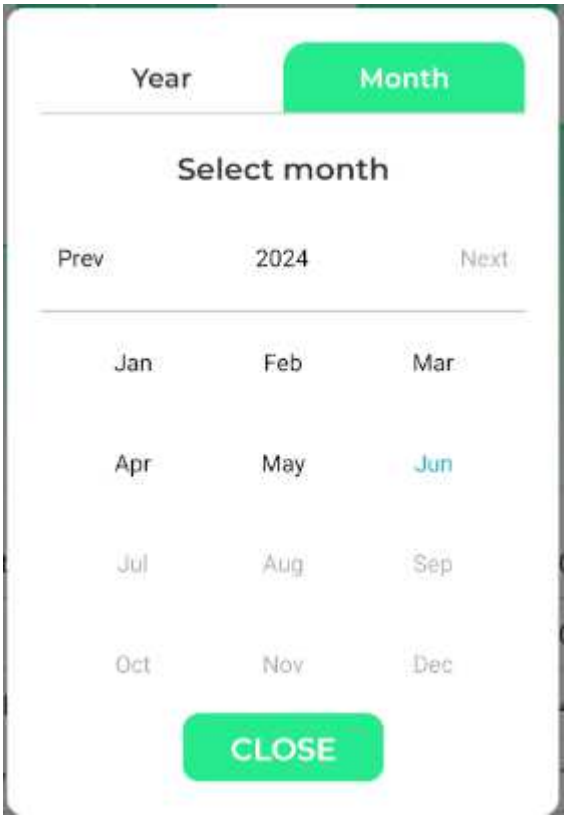
4.4.2. History

Ekran „History“ koji je vidljiv na slici 48. sastoji se od dva dijela – dijagram i popis transakcija. Dijagram prikazuje liniju koja reprezentira kretanje stanja odabranog računa za odabrano vremensko razdoblje. Na slici je prikazano kretanje računa „Card“ kroz peti mjesec u 2024. godini. Vremensko razdoblje moguće je odabrati dodirom na lijevi gumb pri vrhu stranice koji otvara dijaloški prozor koji je vidljiv na slici 49. Moguće je odabrati željeni mjesec ili čitavu godinu dodirom na dugme „Year“. Račun je moguće odabrati dodirom na dugme koje se nalazi pored dugmeta za odabir vremenskog razdoblja, a dijaloški prozor koji se otvara prikazan je na slici 50.

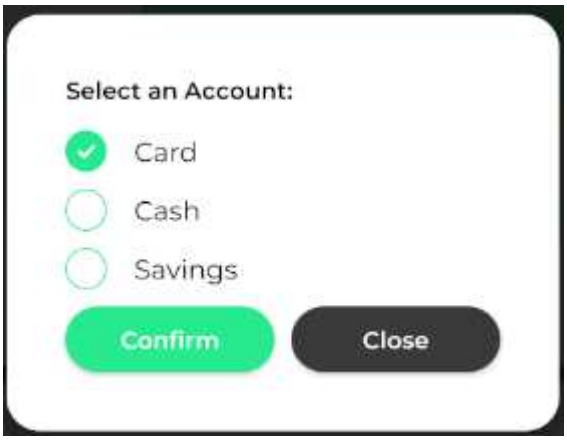


Slika 48. Ekran „History“

Dijagram na x osi prikazuje dane ili mjesece, dok na y osi prikazuje iznose u eurima koji su vrijednosti stanja računa. Ispod samo dijagram nalazi se popis sortiran silazno koji čini deset iznosnom najvećih transakcija za odabrano razdoblje.



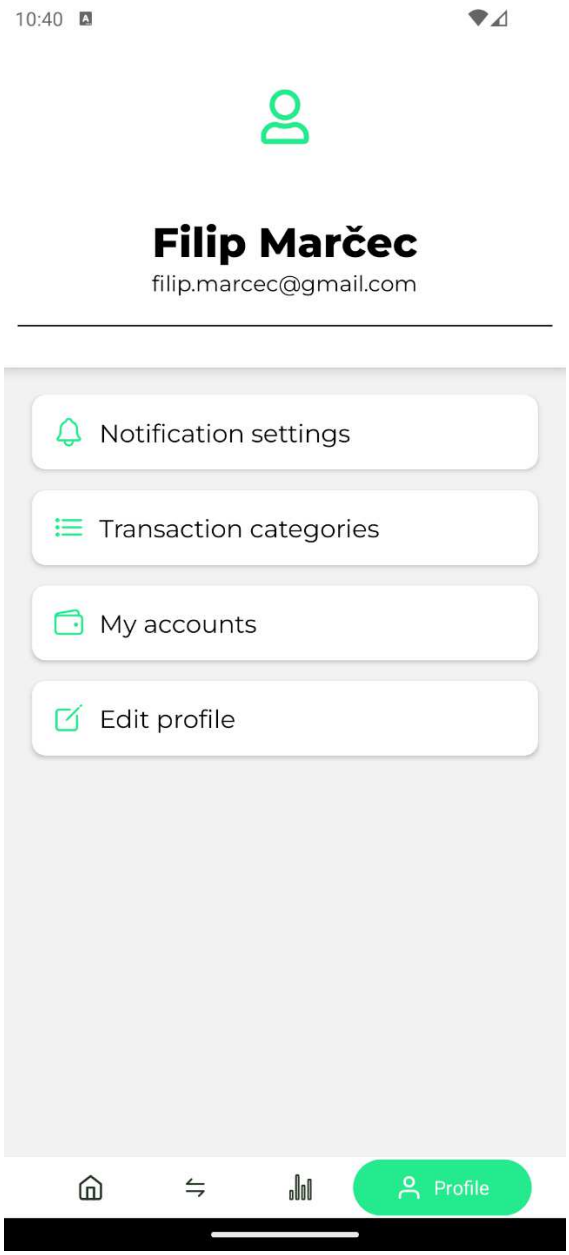
Slika 49. Odabir vremenskog razdoblja



Slika 50. Odabir računa

4.5. Postavke i profil

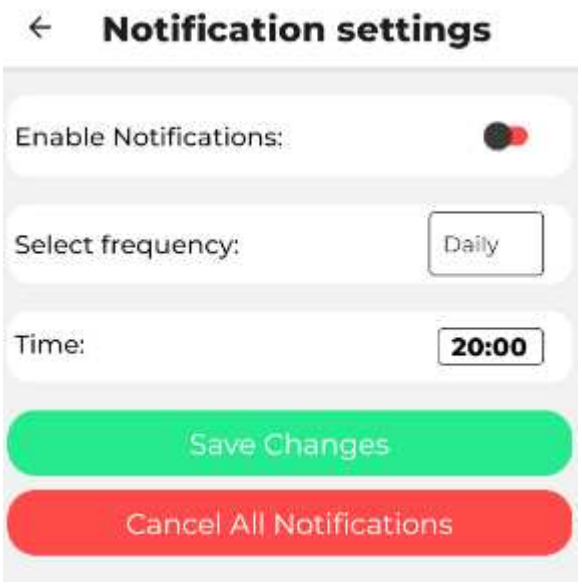
Posljednji dio aplikacije nudi korisniku mogućnosti postavljanja obavijesti, upravljanje kategorijama transakcija, upravljanje stanjem računa te uređivanje osobnih podataka. Svaka od ovih opcija bit će opisana zasebno u nastavku, a opisani ekran vidljiv je na slici 51.



Slika 51. Postavke i profil

4.5.1. Postavke obavijesti

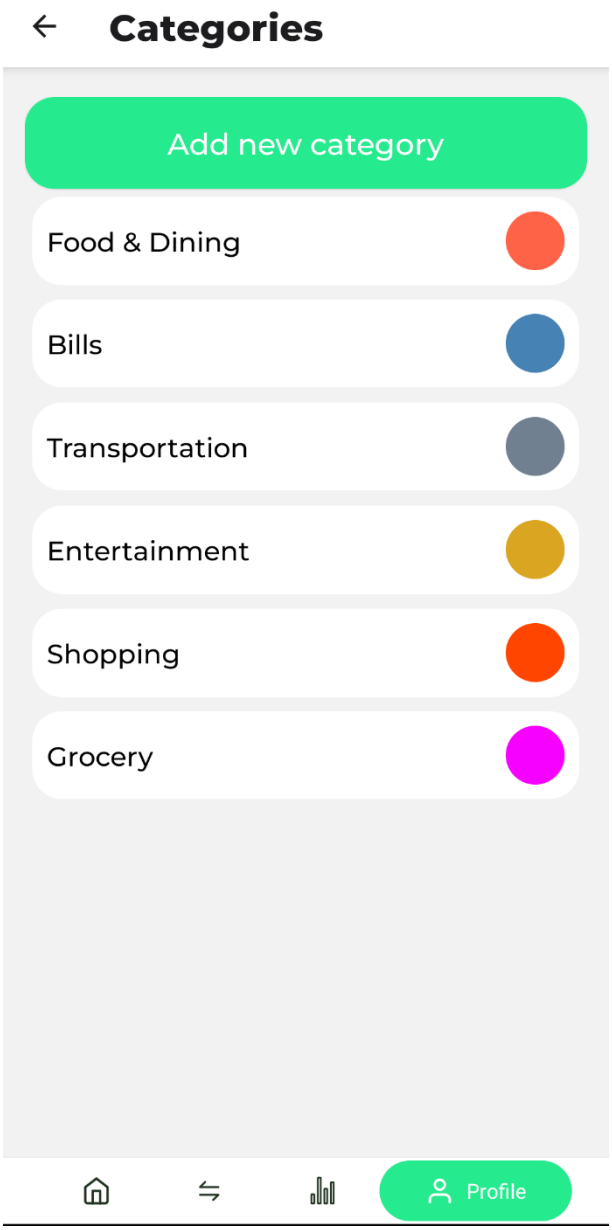
Na slici 52. nalazi se ekran koji omogućuje korisniku da uključi slanje obavijesti od strane aplikacije. Korisnik također ima mogućnost odabrati učestalost slanja obavijesti – dnevno, svaka dva dana, svaka tri dana, tjedno ili svaka dva tjedna, te može odabrati vrijeme u koje želi da mu se ta obavijest pošalje. Postavke se spremaju dodirom na dugme „Save Changes“.



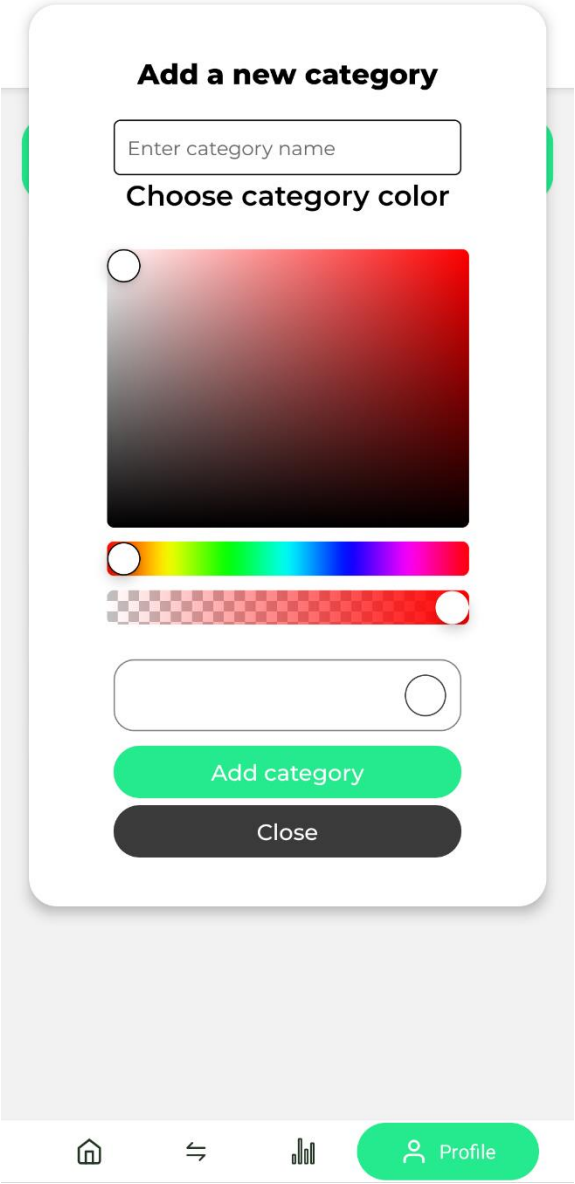
Slika 52. Postavke obavijesti

4.5.2. Upravljanje kategorijama transakcijama

Dodirom na dugme „Transaction categories“ otvara se ekran sa slike 53. koji sadrži popis kategorija transakcija definiranih od strane korisnika. Uz ime svake kategorije nalazi se i boja koju je korisnik odabrao za tu kategoriju. Na slici 54. nalazi se ekran putem kojeg se dodaje nova kategorija. Od korisnika se traži da odabere naziv kategorije te boja koja će tu kategoriju predstavljati. Proces se završava dodirom na dugme „Add Category“. Dodirom na neku od kategorija u popisu kategorija na slici 53. otvara se isti takav ekran samo sa već popunjenim nazivom i odabranom bojom koji omogućuje uređivanje kategorije ili njeno brisanje.



Slika 53. Upravljanje kategorijama

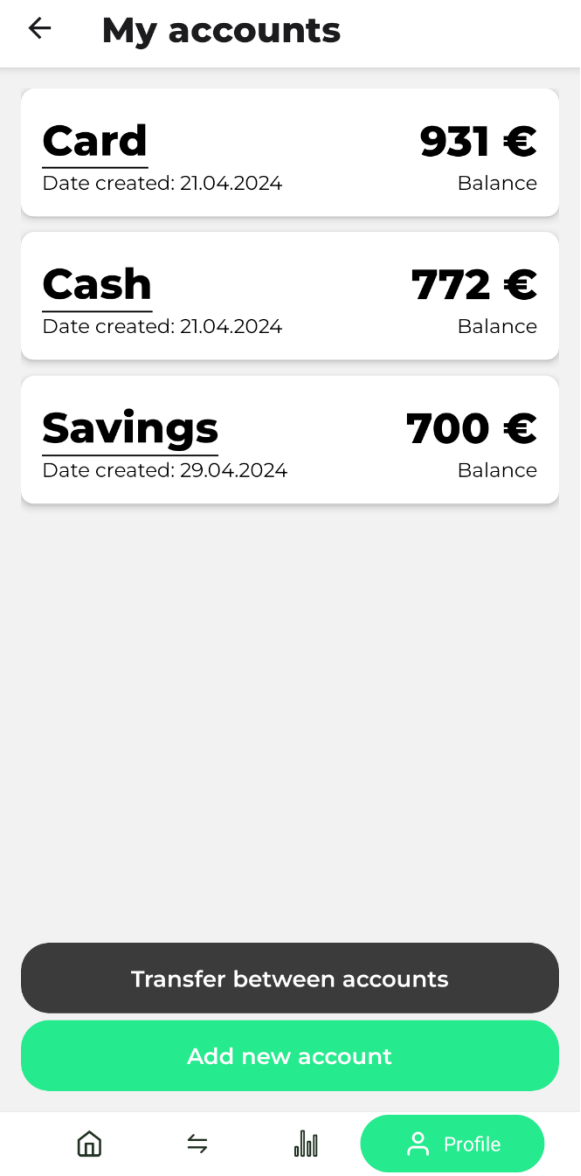


Slika 54. Dodavanje nove kategorije

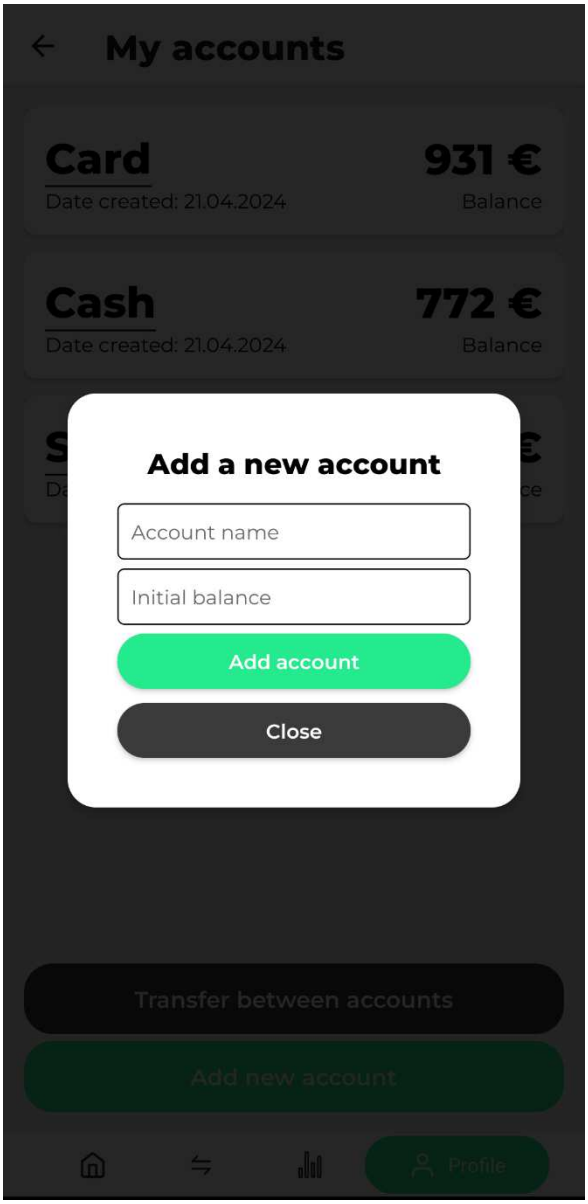
4.5.3. Upravljanje računima

Dodirom na dugme „My accounts“ otvara se ekran sa slike 55. koji sadrži popis korisničkih računa. Svaki račun je prikazan u obliku kartice koja sadrži naziv računa, datum kreiranja i trenutno stanje računa u eurima. Na dnu ekrana nalaze se dva dugmeta: „Transfer between accounts“, koje omogućava prijenos sredstava između računa, i „Add new account“, putem kojeg korisnik može dodati novi račun. Dodavanje novog računa odvija se putem obrasca koji s nalazi na slici 56. Potrebno je unijeti naziv računa i njegovo početno stanje. Slično kao i kod

kategorija transakcija, dodirom na neku karticu računa otvara se isti takav obrazac, samo već popunjen podacima koji se mogu urediti.

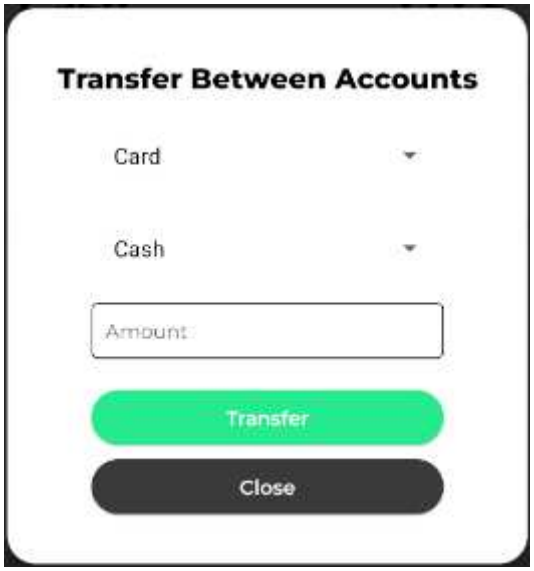


Slika 55. Upravljanje računima



Slika 56. Dodavanje novog računa

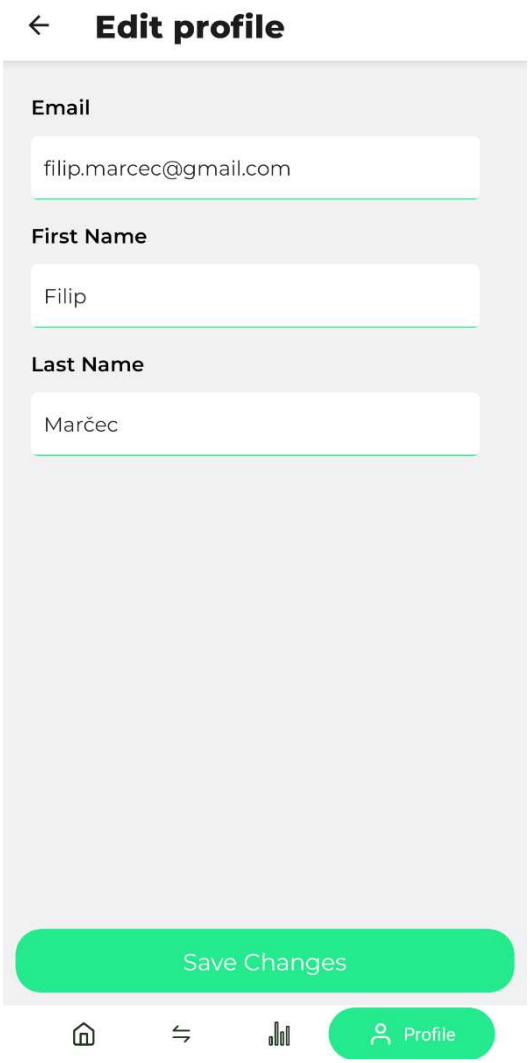
Na slici 57. nalazi se obrazac pomoću kojeg je moguće napraviti prijenos sredstava između dva korisnikova računa. Potrebno je iz dva padajuća izbornika odabrati račun s kojeg se radi prijenos i račun na koji se radi prijenos te iznos koji se želi prenijeti.



Slika 57. Prijenos sredstava na vlastiti račun

4.5.4. Uređivanje osobnih podataka

Na slici 58. nalazi se ekran koji omogućuju korisniku izmjenu osobnih podataka. Moguće je promijeniti email adresu, ime i prezime. Željene promjene spremaju se dodirom na dugme „Save Changes“.



Slika 58. Uređivanje osobnih podataka

5. Korištene tehnologije

5.1. React Native

React Native [4] je JavaScript okvir dizajniran za razvoj mobilnih aplikacija za iOS i Android platforme. Njegova glavna prednost je mogućnost korištenja istog baznog koda za razvoj aplikacija na obje platforme, što znatno smanjuje vrijeme i troškove razvoja. React Native se temelji na principima Reacta [3], popularne JavaScript biblioteke koja se koristi za izradu web aplikacija.

Korištenje React Nativea donosi brojne prednosti, uključujući brži razvoj aplikacija, bolje performanse i fleksibilnost u radu s *nativnim* komponentama uređaja. Osim toga, React Native koristi JSX, što omogućuje kombiniranje JavaScripta i HTML-a unutar iste datoteke. Ovaj pristup pojednostavljuje izgradnju korisničkih sučelja i olakšava održavanje koda.

5.2. Django

Django [7] je web okvir za Python koji omogućuje brzo i učinkovito razvijanje web aplikacija. Razvijen s naglaskom na ponovnu upotrebu koda, brz razvoj i sigurnost, Django pruža sve što je potrebno za izgradnju složenih web aplikacija s minimalnim kodom. Korištenje Django-a u razvoju poslužiteljskog dijela aplikacije osigurava strukturiran i organiziran pristup upravljanju podacima i poslovnom logikom.

Django koristi vlastiti ORM (*Object-Relational Mapping*) što omogućuje razvojnim programerima da umjesto pisanja SQL upita pišu isključivo Python kod. Opisani proces pojednostavljuje upravljanje podacima te povećava produktivnost. Uz to, Django nudi gotova rješenja za autentifikaciju, upravljanje korisnicima i sjednicama što smanjuje rizik od sigurnosnih problema.

5.3. PostgreSQL

PostgreSQL [8] je sustav otvorenog koda (eng. *open-source*) za upravljanje relacijskim bazama podataka poznat po stabilnosti i skalabilnsoti. Omogućuje pohranu i obradu složenih podataka, uključujući JSON, XML i GIS, što ga čini prikladnim za razne vrste aplikacija.

Transakcije u PostgreSQL-u imaju sljedeća svojstva: atomarnost, konzistentnost, izolaciju i trajnost, što osigurava pouzdano upravljanje podacima.

5.4. DataGrip

DataGrip je integrirana razvojna okolina (IDE) za baze podataka, razvijena od strane JetBrainsa. Pruža podršku za razne sustave za upravljanje bazama podataka, uključujući PostgreSQL, MySQL, SQLite i mnoge druge. DataGrip omogućuje jednostavno pisanje i izvršavanje SQL upita, pregledavanje i uređivanje podataka te dizajniranje shema baza podataka.

5.5. Visual Studio Code

Visual Studio Code, poznatiji kao VS Code, je uređivač koda razvijen od strane Microsofta. Podržava razne programske jezike, uključujući JavaScript, Python, Javu i mnoge druge. VS Code pruža bogat skup značajki kao što su automatsko dovršavanje koda, *debugging*, integracija s kontrolom verzija (GITHUB) i proširenja koja dodaju dodatne funkcionalnosti.

5.6. Astah UML

Astah UML [6] je desktop aplikacija dizajnirana za kreiranje UML dijagrama, uključujući dijagrame komponenti, obrazaca uporabe, stanja, aktivnosti i mnoge druge.

5.7. Expo

Expo [5] je okvir dizajniran za izgradnju React Native aplikacija. Sastoji se od alata i usluga koje olakšavaju početak razvoja i testiranje aplikacija. Expo omogućuje brzo pokretanje aplikacija i pregled koda na stvarnom uređaju putem Expo aplikacije, eliminirajući potrebu za emulatorima ili povezivanjem uređaja s računalom. Objavljivanje aplikacija je jednostavno i može se izvršiti s nekoliko naredbi u Expo terminalu.

Zaključak

Razvoj moderne tehnologije logično vodi prema digitalizaciji raznih aspekata svakodnevnog života, uključujući upravljanje osobnim financijama koje su se nekada vodile ručno ili putem tablica. Cilj ovog rada bio je proučiti postojeća rješenja na području mobilnih aplikacija za upravljanje osobnim financijama te izraditi vlastito rješenje. Analiza je pokazala da većina aplikacija slijedi sličan obrazac, ali postoji prostor za unapređenje, posebno u pogledu personalizacije i integracije dodatnih funkcionalnosti.

Mobilna aplikacija razvijena u sklopu ovog rada nudi jednostavno i intuitivno korisničko sučelje. Dizajn sučelja je kreiran s namjerom da bude pristupačan i lako upotrebljiv, pružajući korisnicima osnovne funkcionalnosti za praćenje troškova i prihoda. Podaci su vizualno prikazani kroz nekoliko različitih vrsta dijagrama, a mogućnost stvaranja vlastitih kategorija transakcija donosi sloj personalizacije aplikacije.

U budućem razvoju aplikacije, naglasak bi trebao biti na povećanju skalabilnosti i dodavanju novih funkcionalnosti. Jedna od glavnih funkcionalnosti koja bi bitno poboljšala korisničko iskustvo bila bi integracija automatskog praćenja transakcija koje korisnik napravi s vlastitom kreditnom karticom.

Literatura

[1] Money Manager Expense & Budget, https://www.realbyteapps.com/, 16.06.2024.

[2] Financije – upravitelj novca, https://play.google.com/store/apps/details?id=ru.innim.my_finance&hl=hr, 16.06.2024.

[3] ReactJS, https://reactjs.org/, 29.5.2024.

[4] React Native, https://reactnative.dev/, 29.5.2024.

[5] EXPO, https://expo.dev/, 28.5.2024.

[6] AstahUML, https://astah.net/, 1.6.2024.

[7] Django, https://docs.djangoproject.com/en/5.0/, 1.6.2024.

[8] PostgreSQL, https://www.postgresql.org/docs/, 1.6.2024.

Sažetak

Ovaj rad se bavi analizom postojećih rješenja za upravljanje osobnim financijama te opisom izvedbe i funkcionalnosti razvijene mobilne aplikacije. U prvom dijelu rada opisane su dvije mobilne aplikacije koje se bave upravljanjem financijama, nakon čega su navedeni poslovni, funkcionalni i nefunkcionalni zahtjevi za novu aplikaciju. Slijedi detaljan opis baze podataka i arhitekture korištene pri izradi programskog rješenja. Nakon toga, dan je opis korisničko sučelja aplikacije, upute za korištenje te korištene tehnologije.

Razvijena mobilna aplikacija omogućava korisnicima praćenje troškova, prihoda, upravljanje ponavljajućim transakcijama i vizualni pregled podataka. Aplikacija je razvijena korištenjem JavaScript radnog okvira React Native. Za poslužiteljski aplikacije dio korišten je Django, dok je za bazu podataka korišten PostgreSQL.

Ključne riječi: mobilna aplikacija, React Native, Django, PostgreSQL, financije, troškovi, prihodi, upravljanje novcem

Abstract

This paper analyzes existing solutions for personal finance management and describes the implementation and functionalities of a newly developed mobile application. The first part of the paper discusses two mobile applications focused on finance management, followed by the business, functional, and non-functional requirements for the new application. A detailed description of the database and the architecture used in the development of the software solution is provided. Subsequently, the paper outlines the application's user interface, usage instructions, and the technologies utilized.

The developed mobile application allows users to track expenses, income, manage recurring transactions and look at visual. The application is built using the React Native JavaScript framework. Django was used for the server-side application, while PostgreSQL was used for the database.

Keywords: mobile application, React Native, Django, PostgreSQL, finance, expenses, income, money management