



Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin



Aplikacija za menedžment budžeta
Application for budget management

- Diplomski rad -

Student: Bojer Eduardo Eugen

Broj indeksa: SI 6/17

Smer: Informacione tehnologije -
Softversko inženjerstvo

Zrenjanin, 2022. godine



Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin



Aplikacija za menadžment budžeta
Application for budget management

- Diplomski rad -

Mentor:
Doc. dr Eleonora Brtko

Student: Bojer Eduardo Eugen

Broj indeksa: SI 6/17

Smer: Informacione tehnologije -
Softversko inženjerstvo

Zrenjanin, 2022. godine

Sadržaj

1.	Uvod	5
2.	Mobilne aplikacije	6
3.	Android	6
3.1	Android operativni sistem	6
3.2	Android aplikacije	6
4.	Opis tehnologija	7
4.1	Dart	7
4.2	Flutter programski radni okvir	8
4.3	Slojevi Aplikacije: „Aplikacije za menadžment budžeta“	9
5.	Korišćene tehnologije	10
5.1	Android studio	10
5.2	Flutter	10
5.2.1	Integracija Fluttera i Firebasea	11
5.3	Firebase	11
5.3.2	Firebase baze	11
6.	Modelovanje baze podataka	13
6.1	Dijagram slučajeva korišćenja	13
6.2	Dijagram sekvenci	14
6.3	Dijagram aktivnosti	15
6.4	Dijagram komponenti	16
7.	Opis „Aplikacije za menadžment budžeta“	17
8.	Korisničko uputstvo	18
8.1	Login stranica	18
8.2	Registrowanje na aplikaciju	19
8.3	Početna stranica	20
8.4	Izbor tipa transakcije	21
8.5	Registracija priliva ili rashoda	22
8.6	Kategorizacija rashoda	24
9.	Opis implementacije	25
9.1	Dekorator dizajn patern	25
9.2	Servisi, biblioteke i modeli	25
9.3	Interfejs i njegovi gradivni blokovi	31
10.	Zaključak	39

11.	Literatura.....	40
11.1	Knjige i zbornik	40
11.2	Internet stranice	40

1. Uvod

Tokom dosadašnjih studija imao sam priliku da se upoznam sa različitim sferama programerskog rada, a sebe sam pronašao upravo u mobilnom programiranju. Otkriveno interesovanje sam usavršavao i u slobodno vreme putem neformalnog obrazovanja u vidu različitih: predavanja i radionica.

Ljudi današnjice su u svoju svakodnevnu rutinu uveli i korišćenje mobilnih aplikacija. Na samom početku, većina populacije ih možda nije smatrala značajnim dok se danas sve više oslanjaju na njih. Mobilne aplikacije su prepoznate kao dostupni alat za olakšavanje svakodnevnog života svakog čoveka.

Nekada su ljudi u sveskama ili čak knjigama pisali svoje kućne budžete te o njima vodili računa. Danas, dodatnu pomoć baš za taj svakodnevni zadatak nude razne mobilne aplikacije, a između njih i „*Aplikacija za menadžment budžeta*“.

Cilj pravljenja same mobilne aplikacije je pomoć u lakšem vođenju individualnih finansija kao i mapiranje: prihoda i rashoda. Aplikacija će u skorijoj budućnosti biti doradana tako da bude sigurnija za same korisnike, a takođe će jedna od novina biti i povezivanje naloga aplikacije sa bankovnim računom korisnika.

U daljem tekstu su detaljnije opisane: mobilna aplikacija „*Aplikacija za menadžment budžeta*“, programski jezik i okvir, arhitektura, pristup za korisnike, opis implementacije.

„*Aplikacija za menadžment budžeta*“ konstruisana je u radnom okviru pod nazivom Flater (eng. *Flutter*) u Dart programskom jeziku dostupna samo na Android operativnom sistemu.

2. Mobilne aplikacije

Mobilne aplikacije predstavljaju programe koji su konstruisani radi upotrebe na mobilnim uređajima poput: mobilnog telefona, pametnog sata i tableta i one imaju pristup svim funkcionalnostima uređaja. Prema pristupu koji je korišćen u konstruisanju aplikacije delimo ih na:

- Izvorne (*eng. native*) mobilne aplikacije,
- Kvazi izvorne mobilne aplikacije,
- Hibridne mobilne aplikacije i
- Web aplikacije koje se mogu tretirati kao mobilne ukoliko se njima pristupa sa nekog od mobilnih uređaja

„Aplikacija za menadžment budžeta“ pripada grupi izvornih aplikacija, i ako su tehnologije koje su korišćene pri njenoj izradi podobne za funkcionisanje na više platformi, ali za sada aplikacija funkcioniše na jednom operativnom sistemu – Androidu.

3. Android

3.1 Android operativni sistem

Android operativni sistem je namenjen mobilnim uređajima. Razvoj ovog operativnog sistema počeo je 2003. godine, a na tržištu se pojavio 2007. godine kada su na tržištu već postojali: Nokia (Symbian), BlackBerry (RIM), IOS (Apple) kao vodeći. Android je postigao konstantan rast na tržištu što je dovelo do nastavka samog razvoja operativnog sistema. Danas je ovaj operativni sistem na svojoj trinaestoj (XIII) verziji i nastavlja svoj razvoj.



Slika 1 – IOS, Android, RIM i Symbian operativni sistemi

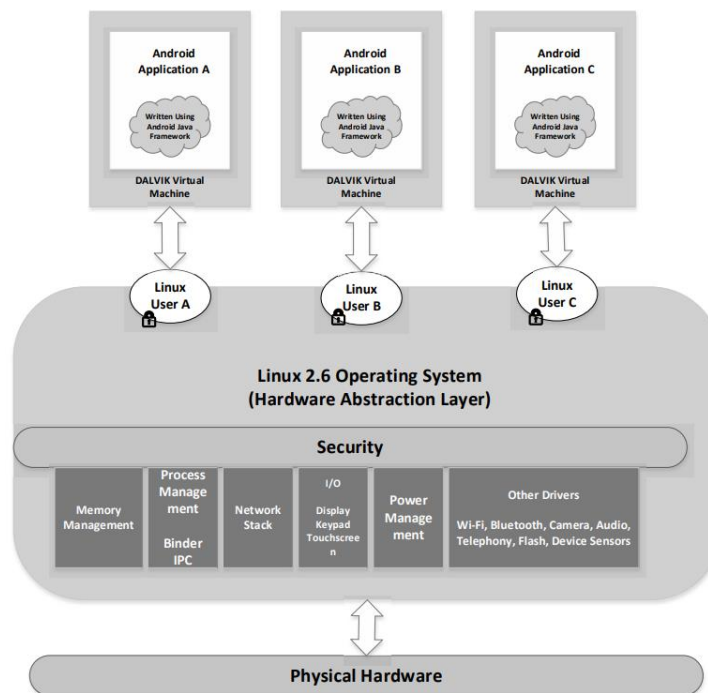
3.2 Android aplikacije

Ne tako davnoj prošlosti počele su da se razvijaju Android aplikacije koje se razvijaju i dan danas. Android aplikacije se realizuju u svom Linuks (*eng. Linux*) procesu zaštićene sigurnosnim funkcijama operativnog sistema. Operativni sistem koristi princip „najmanje

privilegije“ što znači da aplikacija pristupa funkcijama uređaja ukoliko je dobila odobrenje od korisnika.

Kako sve aplikacije Android operativnog sistema tako i: „*Aplikacija za menadžment budžeta*“ ima sledeće osnovne komponente:

- Aktivnosti
- Servise
- Prijemnike obaveštenja
- Komponente koje pružaju sadržaj



Slika 2 – Realizacija aplikacije u Linuks procesima

4. Opis tehnologija

4.1 Dart

Dart je programski jezik koji se pojavio prvi put 2013. godine napravljen od strane Gugla (eng. Google). Na samom početku koristili su ga interni timovi kompanije koja ga je proizvela dok se na tržištu nije pojavila novija verzija (Dart 2.0) pet godina nakon izlaska samog jezika. Druga verzija je dostigla veliki uspeh i postala popularna u programerskom svetu.

Ovaj programski jezik je otvorenog koda (*eng. open source*) te pruža mogućnost rada na mnogo kompleksnijim projektima adaptiranih za uslove modernog internet okruženja. Dart programski jezik ima mnogo alata poput: Darteditor, Dartboard, SDK i mnoge druge.

Iz ovog programskog jezika nastao je programski radni okvir (*eng. framework*) pod nazivom Flater (*eng. Flutter*).

4.2 Flater programski radni okvir

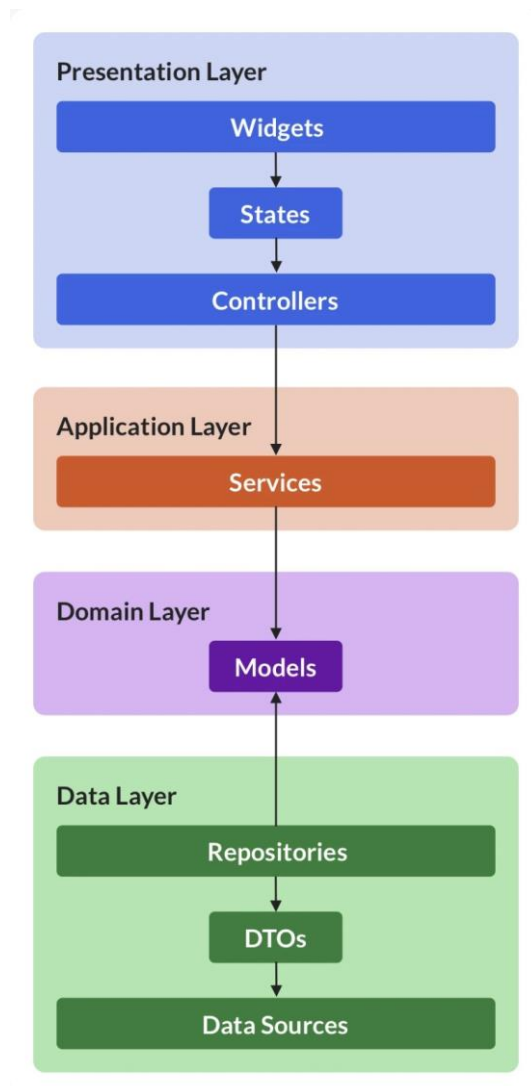
Flater je deo softverske razvojne opreme konstruisane od strane Gugla kroz Dart, programski jezik. Prvi put se na tržištu pojavio 2015. godine kada je uopšteno predstavljen, a za korisnike na internetu bio je dostupan tek dve godine kasnije, 2017. godine.

Skaj (*eng. Sky*) je poznatija prva verzija Flatera koja je radila isključivo na Android operativnom sistemu, dok danas Flater pokriva gotovo sve operativne sisteme. Putem ove alatke moguće je konstruisati razne aplikacije, i web kao i mobilne koje podržavaju svi operativni sistemi i uređaji.



Slika 3 – Ilustracija programskog okvira Flater

4.3 Slojevi Aplikacije: „Aplikacije za menadžment budžeta“



Slika 4- Slojevi „Aplikacije za menadžment budžeta“

Na slici su prikazani slojevi aplikacije kada se određeni podaci pozivaju iz baze te je dijagramski predstavljen redosled funkcija unutar samih slojeva. Taj redosled funkcija predstavlja *call flow*. Aplikacija se sastoji iz sledeća četiri sloja:

- Baza podataka – u kojoj su uskladišteni svi podaci
- Domenski sloj – sadrži modele
- Aplikacioni sloj – sadrži biznis logiku i usko sarađuje sa domenskim slojem
- Prezentacioni sloj ili UI (*User Interface*) sloj – sadrži vidžete

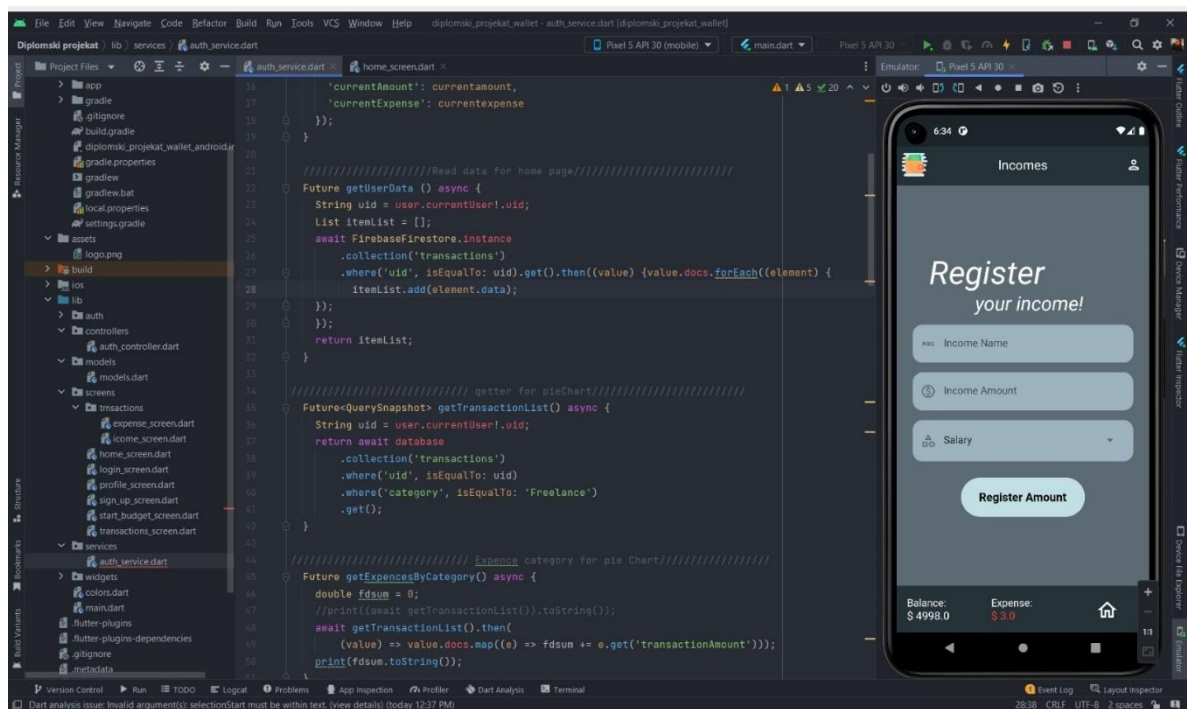
5. Korišćene tehnologije

5.1 Android studio

Android Studio je integrisana okolina za razvoj (*eng. integrated development environment – IDE*) Android aplikacija. Zasnovan je na JetBrainsovom IntelliJ integrisanoj okolini za razvoj softvera i predstavlja službeni IDE za razvoj android aplikacija.

Ova integrisana okolina za razvoj Android aplikacija koristi sistem izgradnje zasnovan na: Gradle-u, emulator, šablone koda kao i Github integraciju. U Android Studiu se kao programski jezik koristi se Java koja je veoma svestran jezik. Upravo zbog svoje svestranosti je spektar u kom se Java koristi jako širok. Takođe, pored Jave koristi se i programski jezik C++.

Android studio ima funkciju pod nazivom: „*Emulator*“. Ova funkcija pruža mogućnost programerima da pokrenu simulaciju Android sistema kako bi mogli da testiraju svoje mobilne aplikacije tokom rada.



Slika 5- Izgled IDE – Android Studia i emulatora

5.2 Flater

Flutter je Guglov radni okvir koji podržava to da jedna aplikacija bude podobna za više operativnih sistema, ali i više uređaja. Namenjen je za konstruisanje: modernih, izvornih i reaktivnih aplikacija. Materijalni dizajn (*eng. Material Design*) predstavlja biblioteku smernica u kojima se nalaze: vidžeti, vizuelne i komponente ponašanja u cilju poboljšanja interfejsa. [1]

Jedna od funkcija ovog radnog je: „Hot reload“ koja omogućava da se u najkraćem mogućem roku osveži pokrenuta aplikacija kada se promeni izvorini kod da bi se dodale određene funkcionalnosti samoj aplikaciji ili da bi se iste modifikovale.

Za potrebe „Aplikacija za menadžment budžeta“ urađena je integracija Flatera sa Firebaseom.

5.2.1 Integracija Flatera i Firebasea

Proces integracije je tekao na sledeći način:

1. Povezivanje google naloga sa Firebase-om
2. Pravljenje projekta na Firebase-u – dodavanje imena projekta posle toga automatski se dobija projektni ID



Slika 6. – Dodavanje projekta na Firebase-u

3. Preuzimanje i implementiranje Firebase skripte u Android projekat

Nakon svih ispunjenih koraka Android projekat će biti dodat u Firebase projekat. Firebase omogućava solucije bez potrebe za serverom, a ujedno se možete osloniti na njega.

5.3 Firebase

Firebase je popularan program koji poznatiji kao “*Backend as a service*” što znači da povezuje backend mobilne i/ili web aplikacije sa bazama podataka i API-a. Lansiran je 2012. godine i dostigao je ogroman uspeh mogućnostima i tehnologijom u samom programu. API unutar Firebasea se sastoji iz SDK koji je kompatibilan za više: platformi i samih programskih jezika

5.3.2 Firebase baze

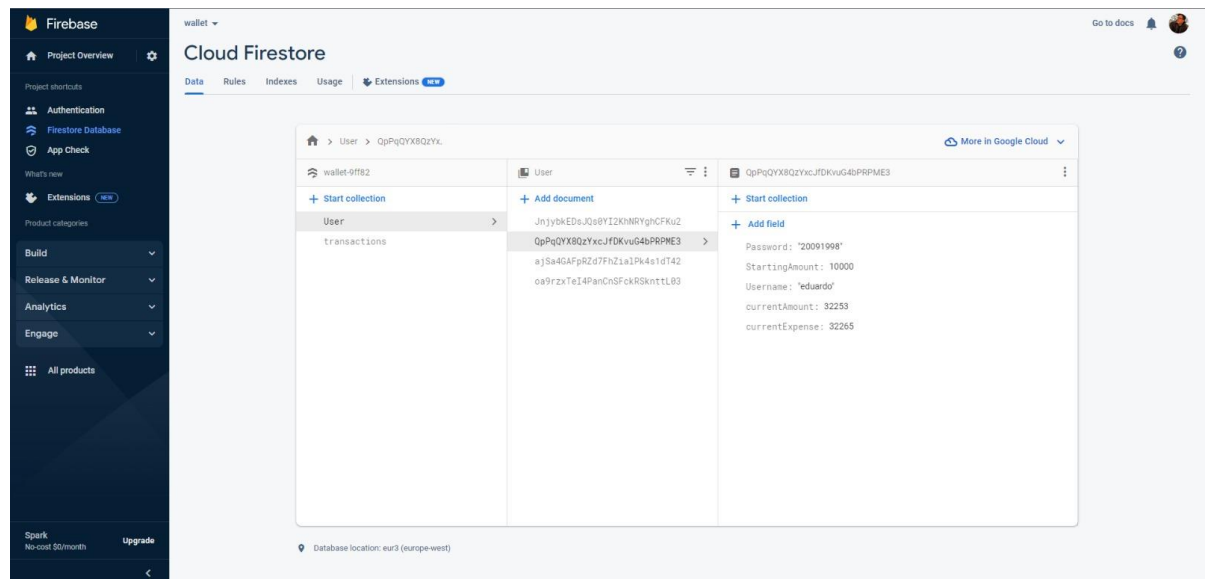
Firebase Store predstavlja bazu izgrađenu putem NoSQL tehnologije, a njen rad se oslanja na modele dokumenata.

Kada govorimo o „Aplikaciji za menadžment budžeta“ postoji jedna Firebase baza. U ovoj bazi sadržane su dve kolekcije:

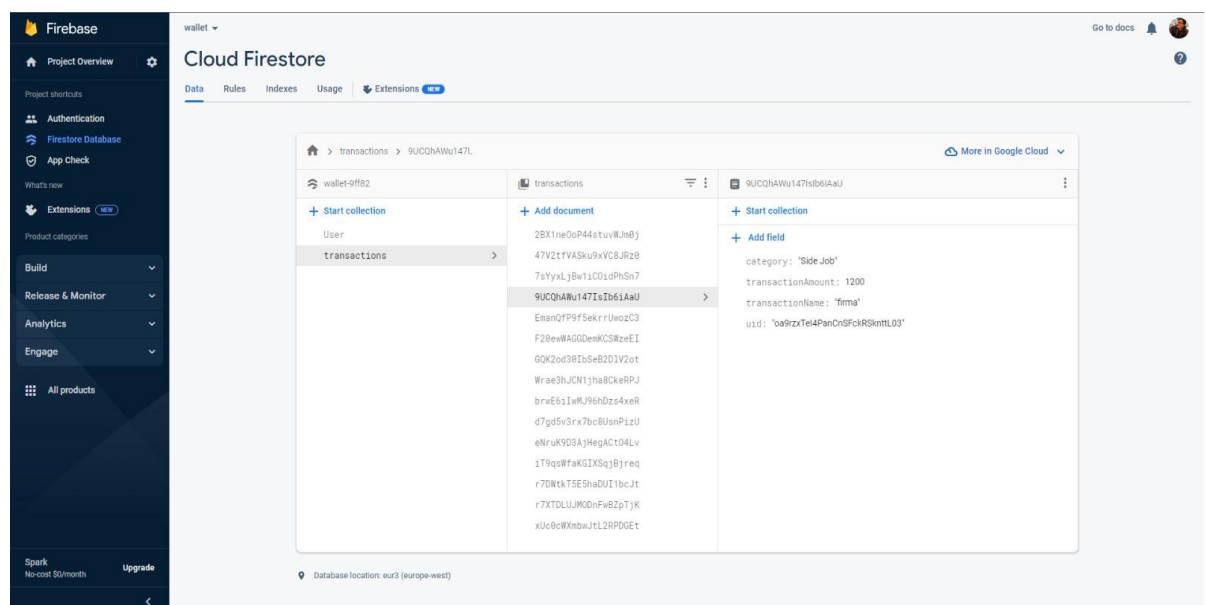
- Korisnik (*eng. user*) i
- Transakcije (*eng. transactions*)

U kolekciji korisnika sakupljeni su svi dokumenti koji sadrže informacije o korisnicima. Informacije su sledeće: ime, početni budžet, trenutni budžet i trenutni rashod. U kolekciji transakcija sakupljeni su svi dokumenti koji sadrže informacije o transakcijama. Informacije su sledeće: kategorija, količina transakcije, ime transakcije i korisnički ID (UID).

Pomoću korisničkog ID-ja povezuju se transakcije i korisnici.



Slika 7 - Kolekcija korisnika



Slika 8- Kolekcija transakcija

Na slikama brojeva sedam i osam se vidi struktura Firebase baze podataka. Podrazumevano je da je baza NoSQL i da funkcioniše po modelima dokumenata te je srazmerna povećanju samog broja korisnika.

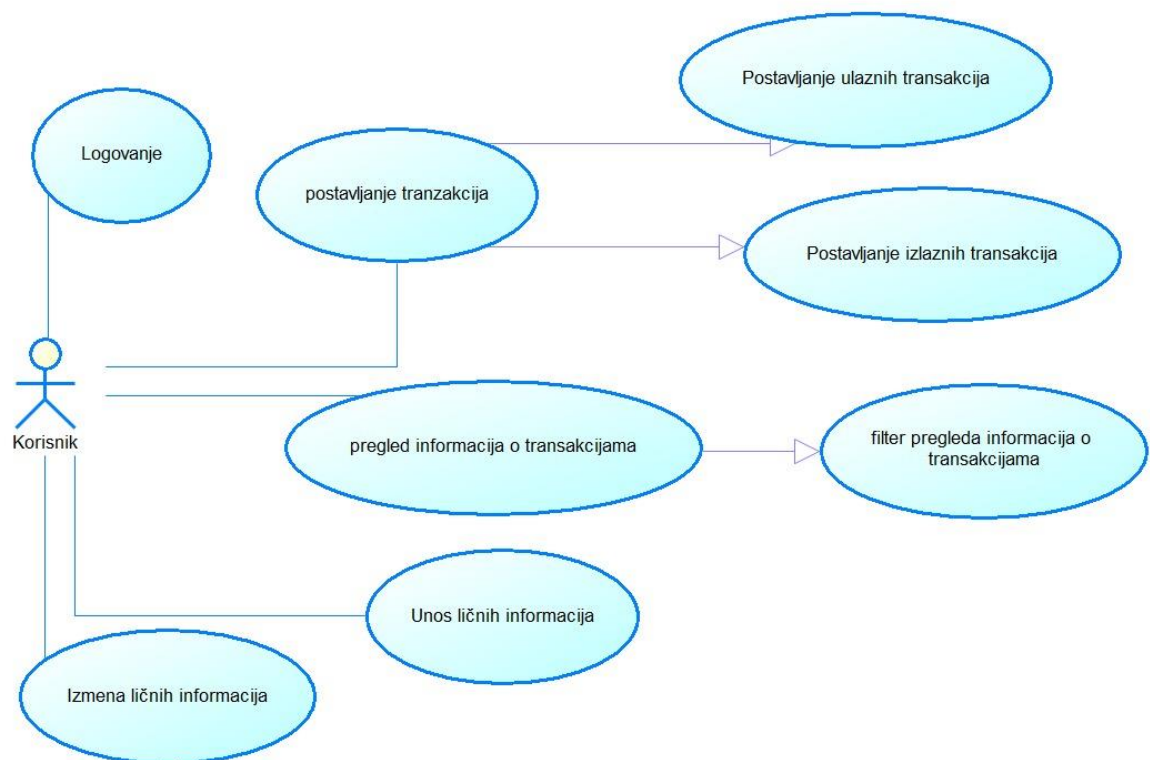
6. Modelovanje baze podataka

Pojavom UML-a i objektno orijentisanog modelovanja dobila se mogućnost pravljenja modela objektno orijentisanih baza podataka, s tim što se na ovaj način razmatra isključivo perzistencija podataka. Zato se i pojavila potreba za modelovanjem relacionih baza podataka koje su rasprostranjenije i popularnije za korišćenje. Ovo je rešeno razvojem UML profila za modelovanje relacionih baza podataka koji definiše proširenja sintakse jezika UML, prilagođavajući je potrebama modelovanja baza podataka [2]. U daljem tekstu opisani su:

- Dijagram slučajeja korišćenja
- Dijagram sekvenci
- Dijagram aktivnosti
- Dijagram komponenti

6.1 Dijagram slučajeja korišćenja

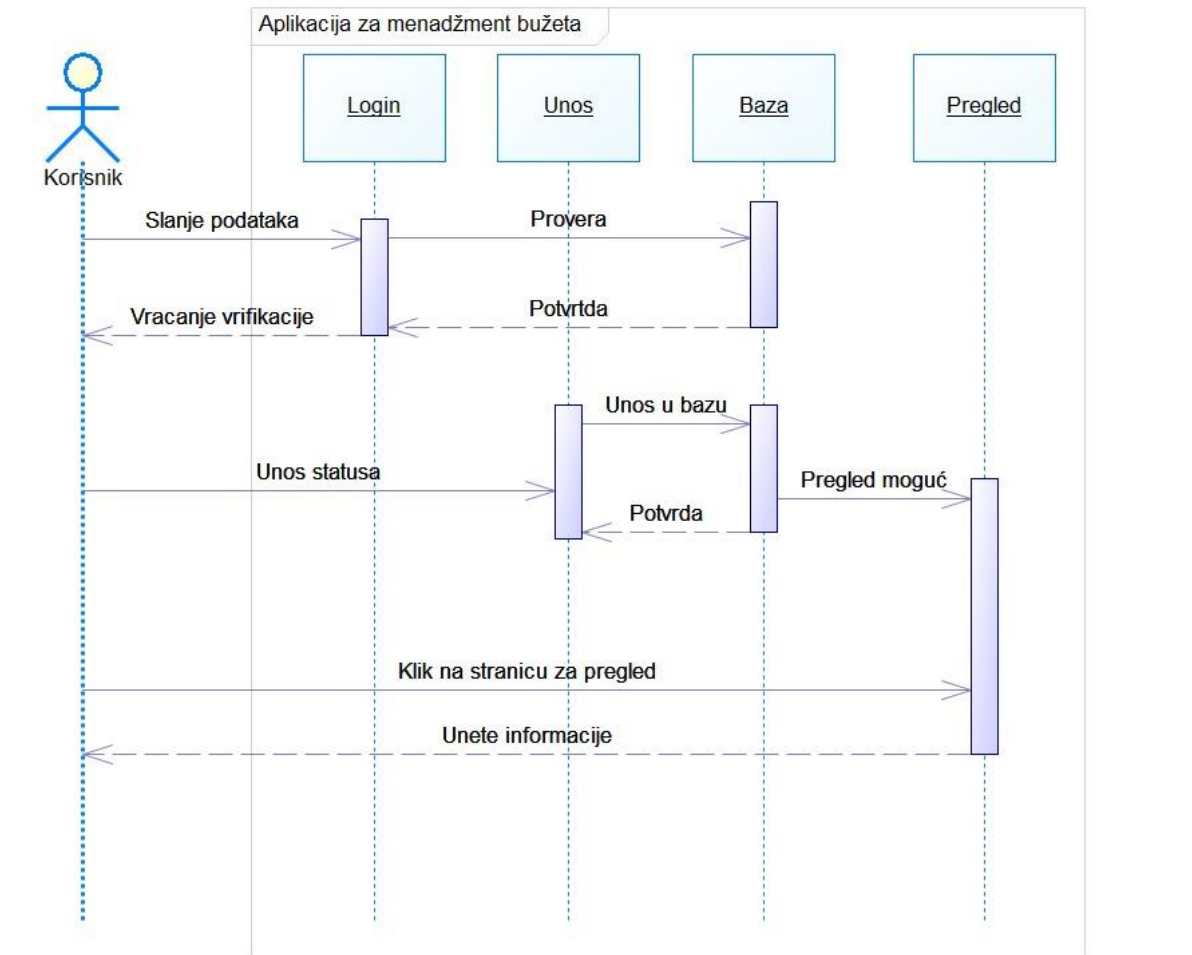
Dijagrami slučajeja korišćenja predstavljaju skup: slučajeja korišćenja, aktera i njihovog međusobnog odnosa. Pseban značaj pridaju u domenu organizovanja i modeliranja sistema jer pored toga što identifikuju korisnika oni i predviđaju odnos korisnika i sistema.



Slika 9- Dijagram slučajeja korišćenja

6.2 Dijagram sekvenci

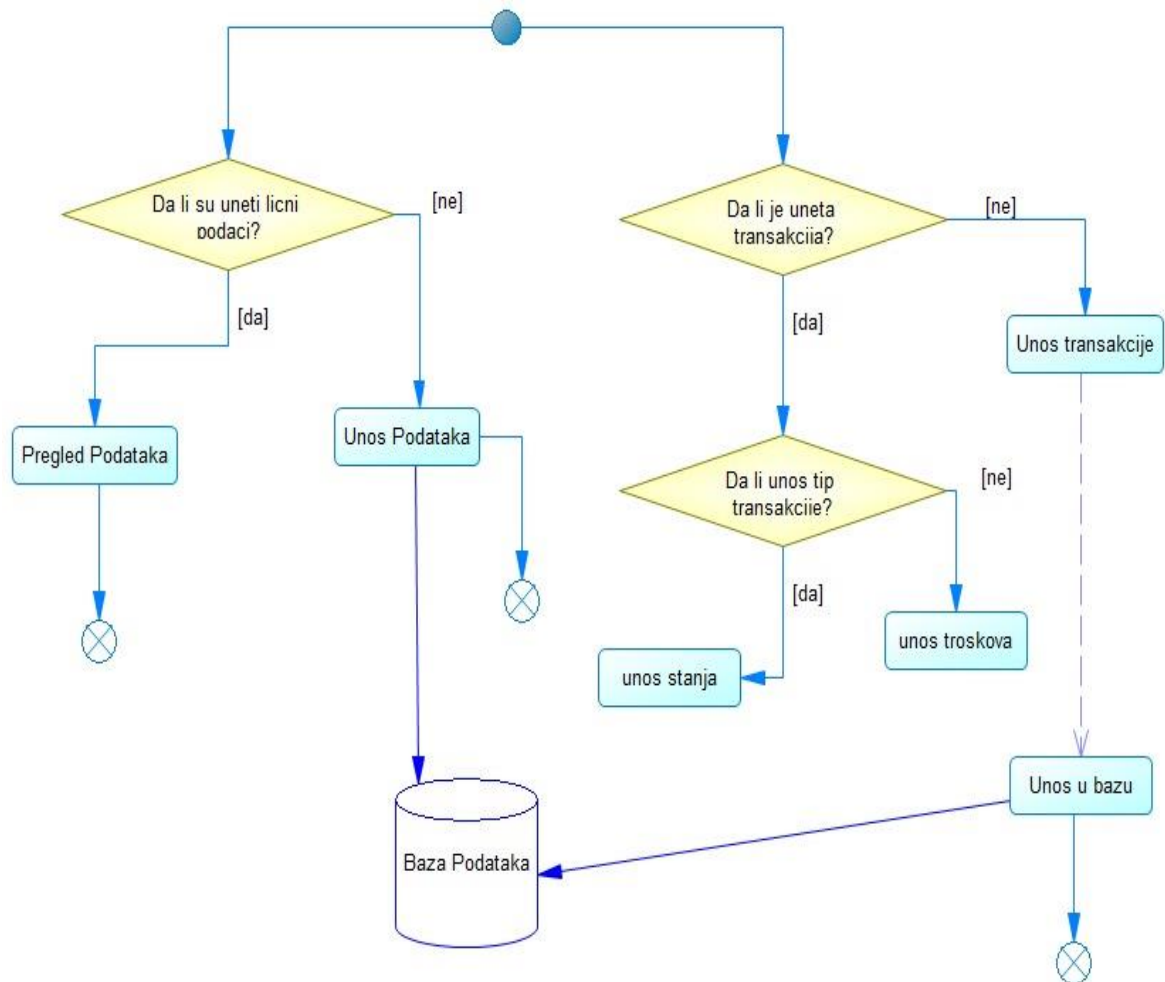
Dijagram sekvence prikazuje interakciju koja modeluje jedan scenario koji se dešava u sistemu. Uz dijagrame klase, najčešće korišćeni UML dijagrami.



Slika 10- Dijagram sekvenci

6.3 Dijagram aktivnosti

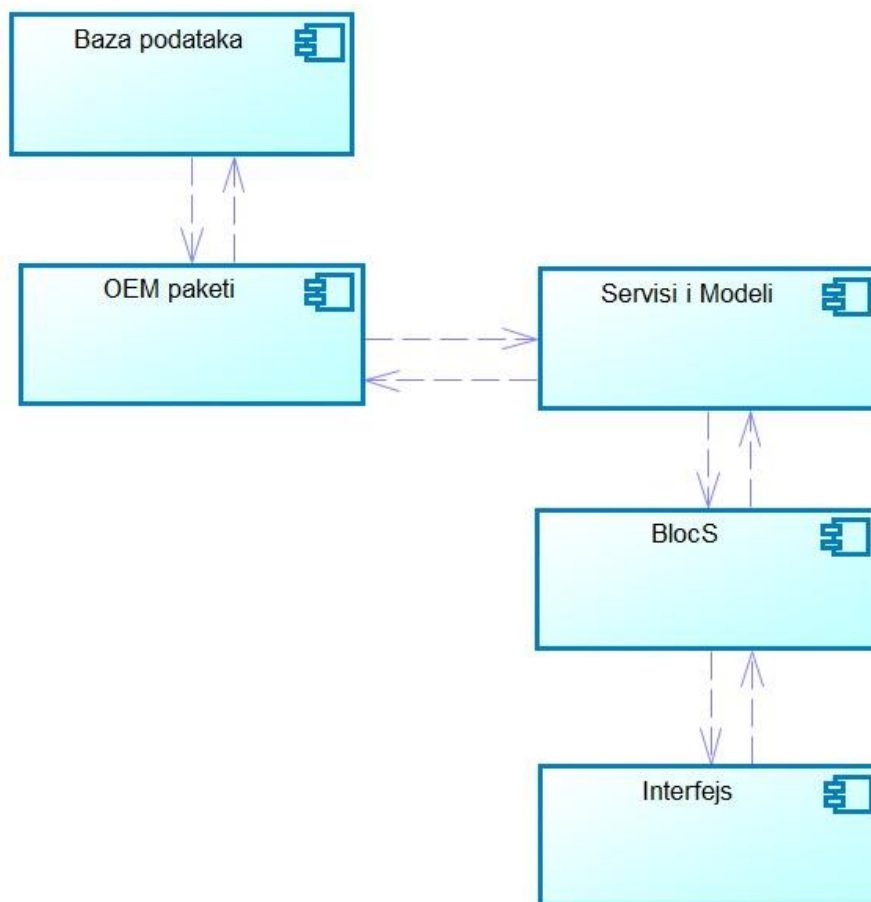
Dijagram aktivnosti predstavlja jednu od varijanti dijagrama stanja. U njemu je opisan tok od jedne do druge aktivnosti sa naglaskom na tok kontrole između objekata.



Slika 11- Dijagram aktivnosti

6.4 Dijagram komponenti

Dijagrami komponenti bliže opisuju same odnose u skupu komponenti, a pružaju statički pregled implementacije. Komponenta obično implementira jednu ili više klasa ili interfejsa.



Slika 12- Dijagram komponenti

7. Opis „Aplikacije za menadžment budžeta“

Mobilna aplikacija: „*Aplikacije za menadžment budžeta*“ za cilj ima da prati stanje budžeta pojedinca, ali i da kategoriše i mapira prihode i rashode. Takođe, aplikacija poseduje statistički prikaz u vidu kružnog dijagrama na kom su prikazani rashodi. Korisniku je omogućeno da sa više uređaja pristupi svom nalogu, a preduslovi su da na uređaju postoji ta aplikacija, ali i da uređaj koristi Android operativni sistem.

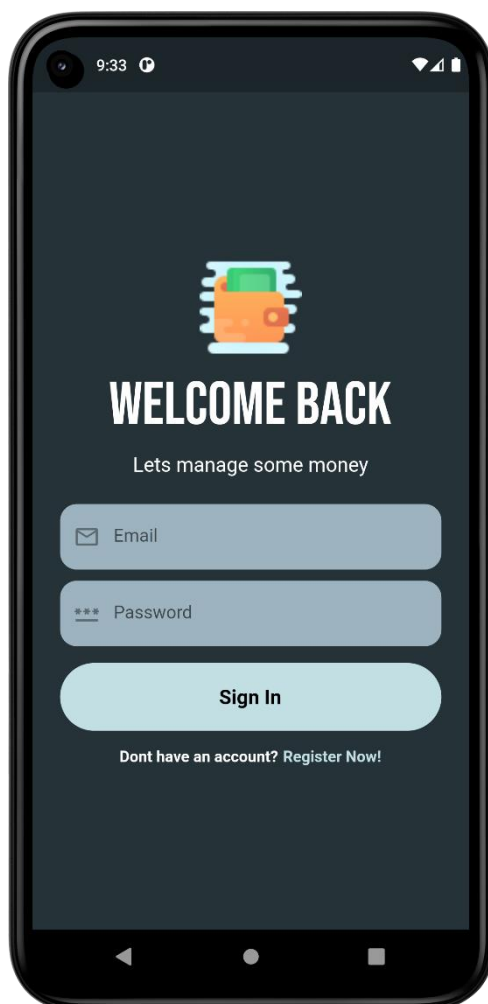
Korisnik pristupa aplikaciji potrebno je da unese email i šifru kako bi pristupio svom nalogu. Pristupom na nalog korisniku su dostupne sve potrebne informacije koje se tiču njegovog budžeta.

Jedan od najvećih problema današnjeg čoveka je da identifikuje “kradljivce” novca, a ova aplikacija nudi rešenje za ovaj problem. Za skoro svaki životni aspekt postoji određena kategorija u aplikaciji gde korisnik može da unese koliko je novca potrošio na tu kategoriju.

Aplikacija je namenjena za korišćenje od strane ljudi koji bi voleli da prate svoj budžet na internetu.

8. Korisničko uputstvo

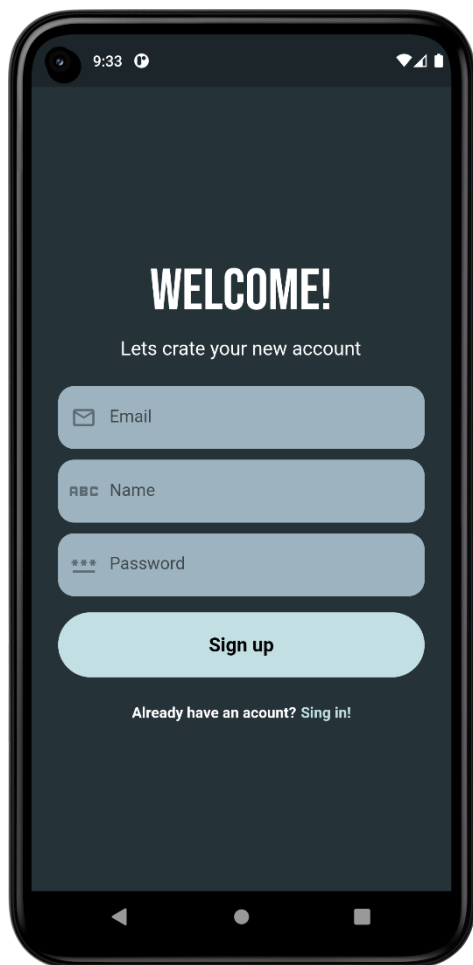
8.1 Login stranica



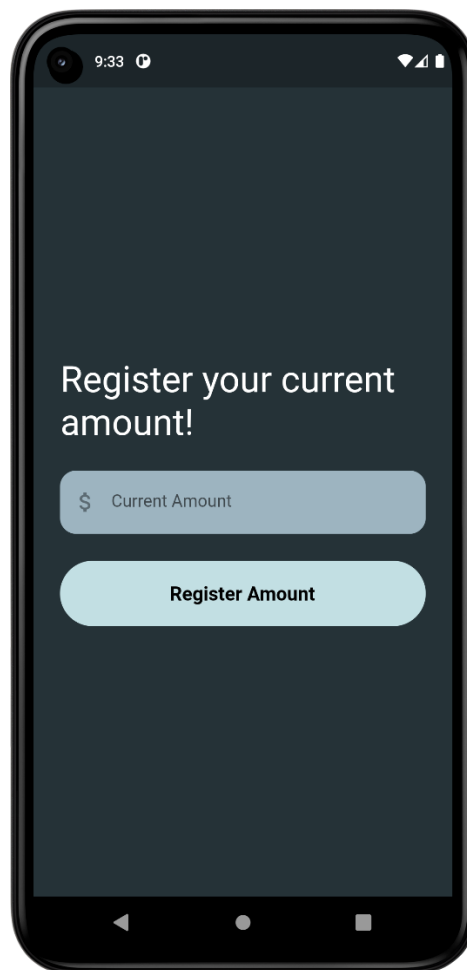
Slika 13- Login stranica

Kada korisnik pristupi aplikaciji prvo što je potrebno da uradi je da se uloguje tako što će uneti svoju email adresu i šifru. Ukoliko korisnik idalje nema nalog na aplikaciji potrebno je da klikne "Register Now", tekst registrator.

8.2 Registrovanje na aplikaciju



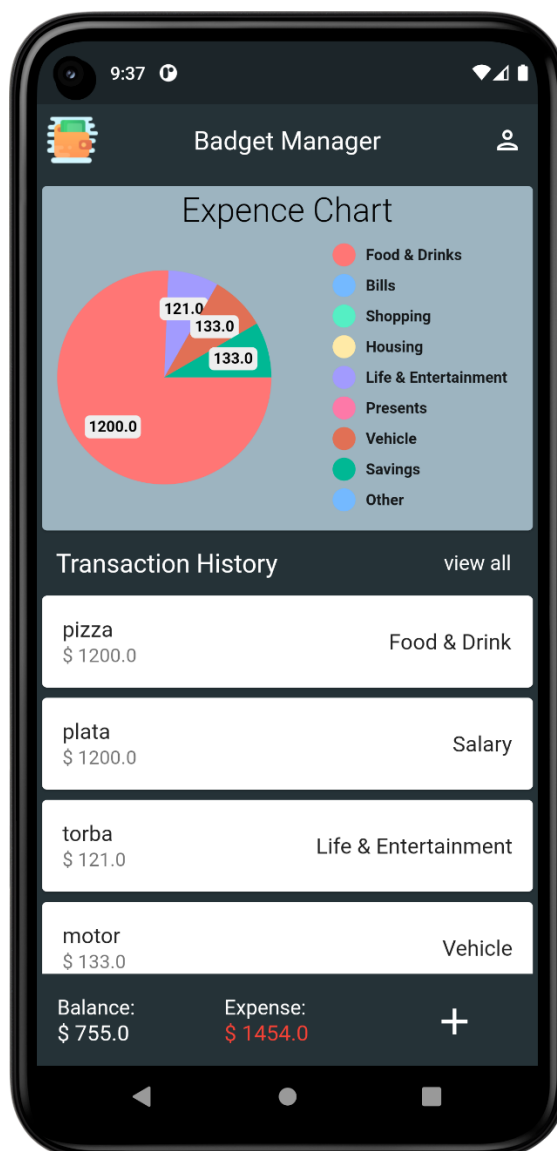
Slika 14- Registracija Korisnika



Slika 15 – Registracija budžeta

Ukoliko neko želi da postane korisnik potrebno je da unese sledeće podatke: email, ime i šifru. Potrebno je da pritisne dugme “Sing up” što će ga preusmeriti na sledeću stranicu, “Starting budget“. Na ovoj stranici upisuje svoje trenutno materijalno stanje i potrebno je da klikne na dugme “Register Amount“, te se nalog registruje na samoj aplikaciji kao i početni budžet. Korisnika zatim uloguje i vodi ga na početnu stranicu.

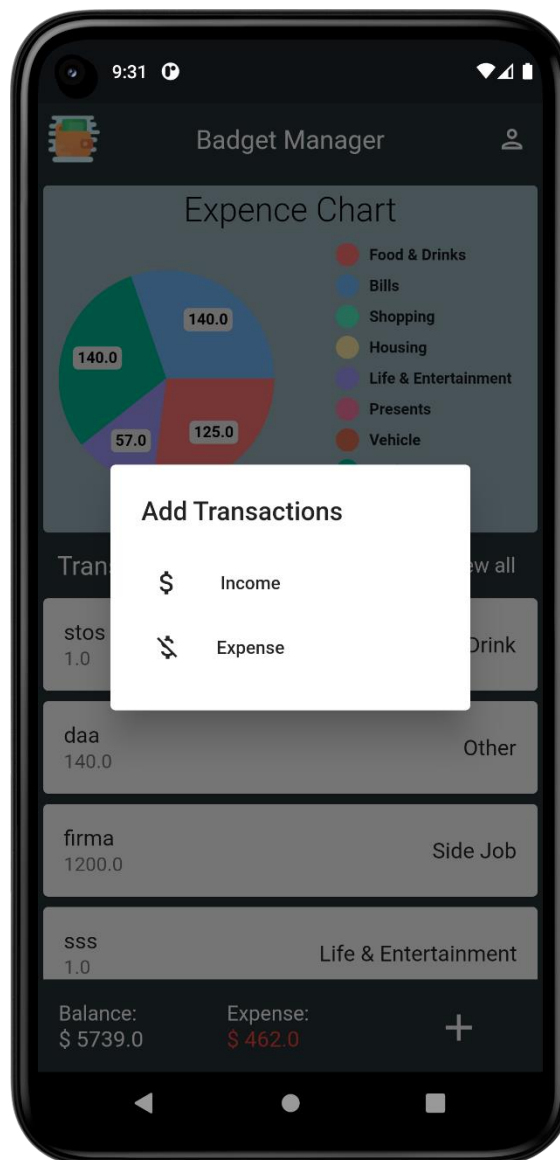
8.3 Početna stranica



Slika 16- Početna stranica

Sa donje strane ekrana možemo videti informacionu traku na kom su izlistani trenutno stanje i troškovi dok je plus dugme koje vodi korisnika na meni za dodavanje priliva i rashoda novca. Odmah iznad informacione trake nalazi se lista svih transakcija. Ona se sastoji iz: informacionog dela koji čine izlistane kategorije i "view all" dugmeta koje vodi korisnika na stranicu o kategorijama rashoda. Nakon toga je prikaz statistike svih rashoda u bazi putem kružnog dijagrama.

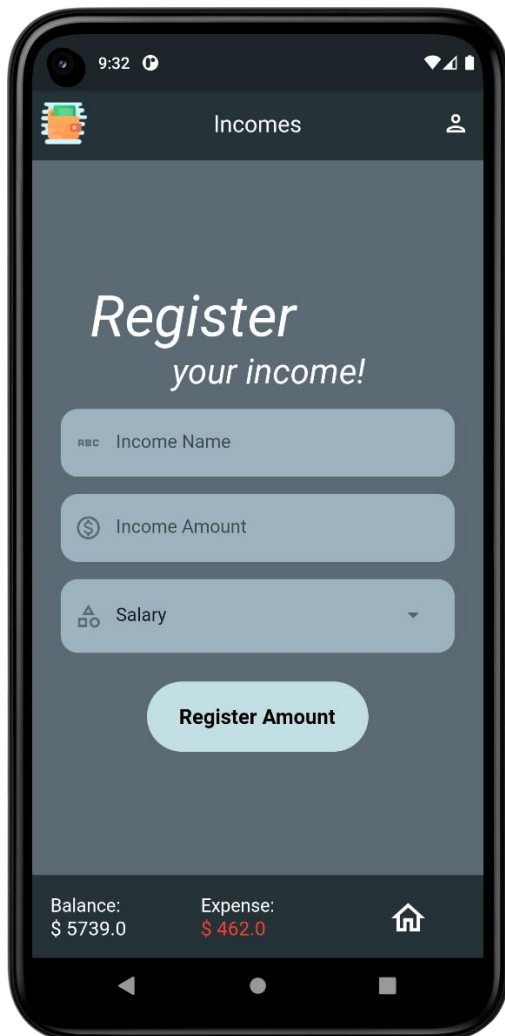
8.4 Izbor tipa transakcije



Slika 17- Meni za izbor tipa transakcije

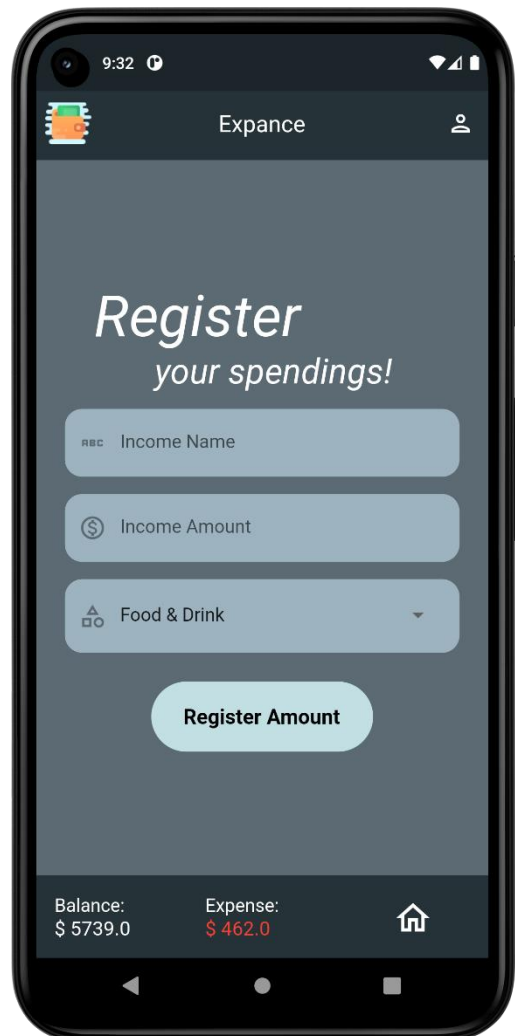
Posle klika na dugme “plus“ informacione trake korisniku se pojavljuje prozor u kom može da izabere kakav tip troškova želi pritiskom na jednu od te dve kategorije te će ga shodno izboru preusmeriti na sledeću stranicu.

8.5 Registracija priliva ili rashoda



The screenshot shows a mobile app interface for recording income. At the top, the status bar displays the time 9:32 and battery level. The app header is titled 'Incomes' with a user icon on the right. The main heading is 'Register your income!'. Below this are three input fields: 'Income Name' (with a small 'RBC' icon), 'Income Amount' (with a dollar sign icon), and a category dropdown menu currently set to 'Salary'. A light blue 'Register Amount' button is positioned below the fields. At the bottom, a summary bar shows 'Balance: \$ 5739.0' and 'Expense: \$ 462.0' in red, with a home icon on the right.

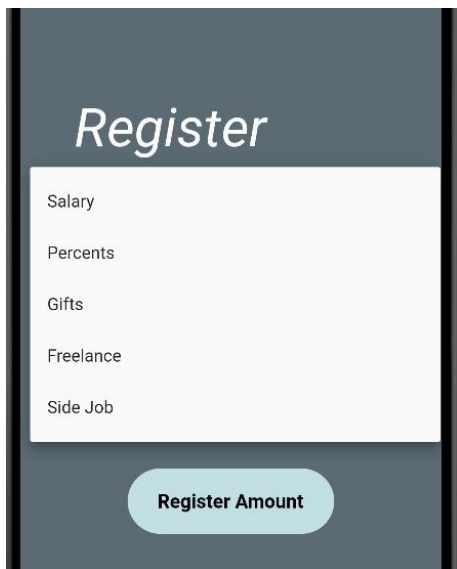
Slika 18 - Unošenje priliva



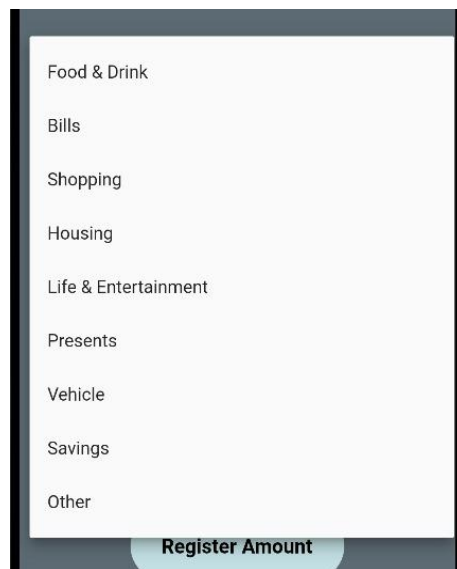
The screenshot shows a mobile app interface for recording expenses. At the top, the status bar displays the time 9:32 and battery level. The app header is titled 'Expense' with a user icon on the right. The main heading is 'Register your spendings!'. Below this are three input fields: 'Income Name' (with a small 'RBC' icon), 'Income Amount' (with a dollar sign icon), and a category dropdown menu currently set to 'Food & Drink'. A light blue 'Register Amount' button is positioned below the fields. At the bottom, a summary bar shows 'Balance: \$ 5739.0' and 'Expense: \$ 462.0' in red, with a home icon on the right.

Slika 19 - Unošenje rashoda

Posle što je odabrao koju vrstu transakcije želi korisnik će biti odveden ili na stranicu na kojoj će morati da upiše rashod ili prihod. Potrebno je da upiše ime transakcije potom njen iznos i za kraj da izabere kategoriju. Da bi korisnik izabrao odgovarajuću kategoriju potrebno je da klikne na treće polje “Salary” ili “Food & Drinks” kada to uradi pojaviće se padajući meni.



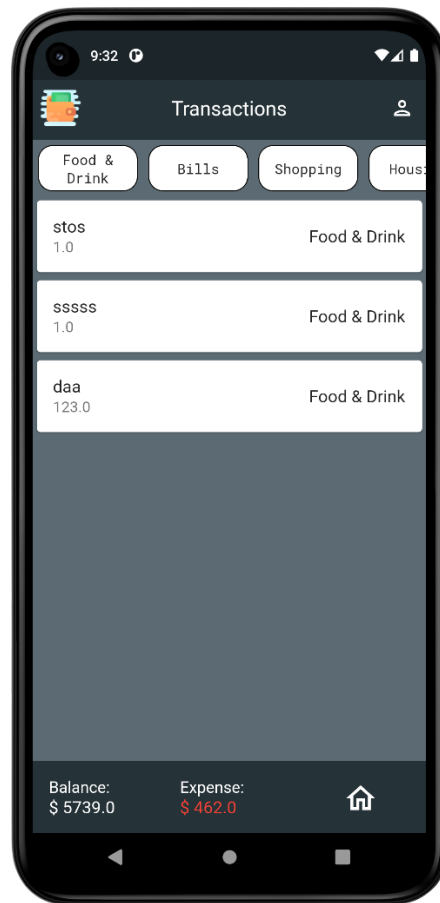
Slika 20 – padajući meni, incom



Slika 21 – padajući meni, spendings

Kada se padajući meniji pojave sve što je potrebo je da korisnik izabere kategoriju pa će ga vratiti na početnu stranicu ,registracije transakcije, i potrebno je da klikne dugme “*Register Amount*“. Potom će ga vratiti na početnu stranicu, a rashod ili prihod će biti unešen u listu transakcija. Takođe, ažuriraće se i kružni dijagram sa novim podacima ako je u pitanju trošak, ali i vrednosti iz informacione trake.

8.6 Kategorizacija rashoda



Slika 22- Kategorizacija rashoda

Kada korisnik klikne na dugme “viwe all” sa liste svih transakcija na početnoj stranici dovedeće ga na stranicu kategorizacija rashoda. Klikom na jednu od gore navedenih kategorija pojaviće mu se lista svih troškova koji upadaju u tu kategoriju.

9. Opis implementacije

9.1 Dekorator dizajn patern

Dinamički pridružuje dodatne odgovornosti nekom objektu Primer: Funkcionalnost objekta se može proširiti statički nasleđivanjem (*eng. compile time*). Međutim, često je neophodno dinamički dodati nove mogućnosti objektu tokom rada sa objektom (*eng.runtime*). Predstavlja fleksibilnu alternativu mehanizmima nasleđivanja radi proširivanja funkcionalnosti Primer rada sa grafičkim korisničkim interfejsom: Dodavanje novih funkcionalnosti grafičkom prozoru zahteva nasleđivanje klase Window da bi se kreirala klasa ScrollableWindow, tj. da bi se kreirao objekat klase ScrollableWindow. Ali, time je nemoguće početi rad sa osnovnim prozorom i dodati mu funkcionalnost u vreme izvršavanja (*eng. runtime*) kako bi postao prozor sa klizačem. [2]

9.2 Servisi, biblioteke i modeli

```
10. class Transaction {
    String? transactionName;
    String? transactionAmount;
    String? category;

    Transaction(this.transactionName, this.transactionAmount,
this.category);

    Map<String, dynamic> json() => {
        'transactionName': transactionName,
        'transactionAmount': transactionAmount,
        'category': category
    };

    Transaction.fromjson(Map<String, dynamic> json) {
        transactionName = json['transactionName'];
        transactionAmount = json['transactionAmount'];
        category = json['category'];
    }
}
```

Listing 1 – Model transakcija

Model transakcije, ova klasa je svoju namenu pronašla u pomoći pri postavljanju vrednosti u kodu koje se posle šalju u bazu podataka. U kodu se mogu vezati za ID korisnika i čitati kada je potrebno.

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class AuthService {
    final FirebaseAuth user = FirebaseAuth.instance;
    final FirebaseFirestore database = FirebaseFirestore.instance;

    List<String> itemsExpende = [
        'Food & Drink',
        'Bills',
        'Shopping',
        'Housing',
    ]
}
```

```

        'Life & Entertainment',
        'Presents',
        'Vehicle',
        'Savings',
        'Other'
    ];

    void crateUser(String username, String password, double amount,
        double currentamount, double currentexpense) {
        currentamount += amount;
        database.collection('User').doc(user.currentUser!.uid).set({
            'Username': username,
            'Password': password,
            'StartingAmount': amount,
            'currentAmount': currentamount,
            'currentExpense': currentexpense
        });
    }

    Future getUserDataByCategory(category) async {
        String uid = user.currentUser!.uid;
        List itemList = [];
        await FirebaseFirestore.instance
            .collection('transactions')
            .where('uid', isEqualTo: uid)
            .where('category', isEqualTo: category)
            .get()
            .then((value) {
                value.docs.forEach((element) {
                    itemList.add(element);
                });
            });
        return itemList;
    }

    Future getUserData() async {
        String uid = user.currentUser!.uid;
        List itemList = [];
        await FirebaseFirestore.instance
            .collection('transactions')
            .where('uid', isEqualTo: uid)
            .get()
            .then((value) {
                value.docs.forEach((element) {
                    itemList.add(element);
                });
            });
        return itemList;
    }

    fetchDataBaseList() async {
        List<double> sum=[0,0,0,0,0,0,0,0,0,0];
        int index= 0;
        for (var item in itemsExpenct){
            sum[index]= await getmountbycat(item);
            index++;
            print(await getmountbycat(item));
        }
        return sum;
    }

```

```

}

getmountbycat(String category) async{
  double sum=0;
  dynamic result = await getUserDataByCategory(category);
  if (result == null) {
    debugPrint("cant get data");
  } else {
    result.forEach((e) {
      sum+=e['transactionAmount'];
    });
    return sum;}
}

void updateCurrentAmount(newamount) async {
  String uid = user.currentUser!.uid;
  double amount = await getCurrentAmount();
  FirebaseFirestore.instance
    .collection('User')
    .doc(uid)
    .update({'currentAmount': amount + newamount});
}

void updateCurrentExpenditure(newamount) async {
  String uid = user.currentUser!.uid;
  double amount = await getCurrentAmount();
  FirebaseFirestore.instance
    .collection('User')
    .doc(uid)
    .update({'currentAmount': amount - newamount});
  double expense = await getCurrentExpense();
  FirebaseFirestore.instance
    .collection('User')
    .doc(uid)
    .update({'currentExpense': expense + newamount});
}

Future getCurrentAmount() async {
  String uid = user.currentUser!.uid;
  DocumentSnapshot samount =
    await
FirebaseFirestore.instance.collection('User').doc(uid).get();
  print(samount.get('currentAmount'));
  return samount.get('currentAmount');
}

Future getCurrentExpense() async {
  String uid = user.currentUser!.uid;
  DocumentSnapshot samount =
    await
FirebaseFirestore.instance.collection('User').doc(uid).get();
  print(samount.get('currentExpense'));
  return samount.get('currentExpense');
}

void addIncome(
  double newIncome,

```

```

        String incomeName,
        String category,
    ) {
        String uid = user.currentUser!.uid.toString();
        database.collection('transactions').add({
            'transactionName': incomeName,
            'transactionAmount': newIncome,
            'category': category,
            'uid': uid
        }).then((value) => updateCurrentAmount(newIncome));
    }

    void addExpenditure(
        double newIncome,
        String incomeName,
        String category,
    ) {
        String uid = user.currentUser!.uid.toString();
        database.collection('transactions').add({
            'transactionName': incomeName,
            'transactionAmount': newIncome,
            'category': category,
            'uid': uid
        }).then((value) => updateCurrentExpenditure(newIncome));
    }

    Future signIn(String email, String password) async {
        await FirebaseAuth.instance.signInWithEmailAndPassword(
            email: email,
            password: password,
        );
    }

    Future signUp(String email, String password) async {
        await FirebaseAuth.instance.createUserWithEmailAndPassword(
            email: email,
            password: password,
        );
    }

    Future signOut() async {
        try {
            return FirebaseAuth.instance.signOut();
        } catch (error) {
            print(error.toString());
            return null;
        }
    }
}

```

Listing 2 – Servis autentifikacije i operacija

U ovom fajlu se nalaze metode koje deklarišu autentifikaciju korisnika kroz: prijavljivanje, registracija i odjava sa aplikacije, pravljenje kolekcije korisnika, kreiranje dokumenta o korisniku u Firebase-u i sve ostale operacije koje su potrebne za funkcionalan rad budžeta.

```

class AuthController {
    static final AuthController _authController =
AuthController._internal();
    factory AuthController() {
        return _authController;
    }
    AuthController._internal();
    List<String> items = ['Salary', 'Percents', 'Gifts', 'Freelance',
'Side Job'];
    List<String> itemsExpenditure = [
        'Food & Drink',
        'Bills',
        'Shopping',
        'Housing',
        'Life & Entertainment',
        'Presents',
        'Vehicle',
        'Savings',
        'Other'
    ];

    final emailController = TextEditingController();
    final passwordController = TextEditingController();
    final nameController = TextEditingController();
    final amountController = TextEditingController();
    final double curamount = 0.0;
    final double expencesamount = 0.0;
    final IncomeNameController = TextEditingController();
    final IncomeAmountController = TextEditingController();
    String Categoryoftransation = 'Salary';
    String CategoryoftransationExpenditure = 'Food & Drink';
    final GlobalKey<FormState> key = GlobalKey<FormState>();
    final AuthService service = AuthService();

    void AddIncome() async {
        service.addIncome(double.parse(IncomeAmountController.text),
            IncomeNameController.text, Categoryoftransation);
    }

    void AddExpenditure() async {
        service.addExpenditure(double.parse(IncomeAmountController.text),
            IncomeNameController.text, CategoryoftransationExpenditure);
    }

    void signup() {
        service
            .signUp(emailController.text.trim(),
passwordController.text.trim())
            .then((value) => service.createUser(
                nameController.text.trim(),
                passwordController.text.trim(),
                double.parse(amountController.text),
                double.parse(curamount.toString()),
                double.parse(expencesamount.toString()),
            ));
    }

    void signIn() async {
        service.signIn(emailController.text.trim(),
passwordController.text.trim());}
}

```

Listing 3 – Kontroler autentifikacije

U ovom fajlu se navode svi kontroleri koji se koriste za prikupljanje podataka iz svih formi i polja u kojima se unose podaci. Radi prosleđivanja podataka dalje pozivaju se metode iz servisa autentifikacije. Takođe, neke od operacija koje rade sa budžetom su ozvane u ovom delu koda. Sve metode koje se nalaze u ovom fajlu se pozivaju u fajlovima gde se koriste kontroleri.

```
class ColorConstants {
  static const kPrimaryColor = Color(0xFF253237);
  static const kSecondaryColor = Color(0xFF5C6B73);
  static const kThirdColor = Color(0xFF9DB4C0);
  static const kFortColor = Color(0xFFC2DFE3);
  static const kWhite = Color(0xFFFFFFFF);
  static const kBlack = Color(0xFF000000);
}
```

Listing 4 – Boja

U ovom fajlu su prikazani modeli boja koje su korišćene u aplikaciji.

```
name: googleauth
description: A new Flutter project.

publish_to: "none"

version: 1.0.0+1

environment:
  sdk: ">=2.17.1 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  firebase_core: ^1.22.0
  flutter_signin_button: ^2.0.0
  google_sign_in: ^5.4.1
  firebase_auth: ^3.9.0
  cloud_firestore: ^3.4.8
  google_fonts:
  form_field_validator: ^1.1.0
  pie_chart: ^5.3.2

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^2.0.0

flutter:

  uses-material-design: true

  assets:
    - assets/
```

Listing 5 - pubspec.yaml

Fajl koji se nalazi iznad, se konstruiše za svaki Flater projekat i sadrži da metapodatke koji omogućavaju bolji rad alata programskog okvira Fatera i programskog jezika Dart-a.

9.3 Interfejs i njegovi gradivni blokovi

```
class _MainPageState extends State<MainPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder<User?>(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return HomePage();
          } else {
            return AuthPage();
          }
        }
      )
    );
  }
}
```

Listing 4 – main_page

U main page služi da proveriti da li je korisnik ulogovan u aplikaciju ukoliko jeste vodi ga na “Home page“, a ukoliko nije na “Auth page“.

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: MainPage(),
    );
  }
}
```

Listing 5.- main.dart

Ovo je glavni fajl on se koristi za pokretanje aplikacije, on poziva fajl main_page.dart koja sadrži “MainPage“ klasu.

```
class LoginScreen extends StatefulWidget {
  final VoidCallback showRegisterPage;
  const LoginScreen({Key? key, required this.showRegisterPage})
    : super(key: key);

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final AuthController controller = AuthController();
}
```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: ColorConstants.kPrimaryColor,
    body: SafeArea(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            //icon/Logo
            Image.asset('assets/logo.png', scale: 0.7),
            SizedBox(height: 10),
            //Top section
            Text(
              'Welcome Back',
              style: GoogleFonts.bebasNeue(
                fontSize: 54,
                color: ColorConstants.kWhite,
              ),
            ),
            SizedBox(height: 10),
            Text(
              "Lets manage some money",
              style: TextStyle(
                fontWeight: FontWeight.w400,
                fontSize: 18,
                color: ColorConstants.kWhite),
            ),
            //email field
            SizedBox(height: 25),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 25.0),
              child: TextField(
                keyboardType: TextInputType.emailAddress,
                controller: controller.emailController,
                decoration: InputDecoration(
                  prefixIcon: Icon(Icons.email_outlined,
                    color: ColorConstants.kSecondaryColor),
                  hintText: "Email",
                  filled: true,
                  fillColor: ColorConstants.kThirdColor,
                  border: OutlineInputBorder(
                    borderRadius: BorderRadius.circular(15),
                    borderSide: BorderSide.none,
                  ),
                ),
              ),
            ),
            //password
            SizedBox(height: 10),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 25.0),
              child: TextField(
                controller: controller.passwordController,
                obscureText: true,
                decoration: InputDecoration(
                  prefixIcon: Icon(Icons.password_outlined,
                    color: ColorConstants.kSecondaryColor),
                  hintText: "Password",
                  filled: true,
                  fillColor: ColorConstants.kThirdColor,

```



```

        border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(15),
          borderSide: BorderSide.none,
        )),
      ),
    ),
  //signin
  SizedBox(height: 15),
  Padding(
    padding: const EdgeInsets.symmetric(horizontal: 25.0),
    child: GestureDetector(
      onTap: () {
        controller.signIn();
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => HomePage()),
        ),
      ),
    child: Container(
      padding: EdgeInsets.all(20),
      decoration: BoxDecoration(
        color: ColorConstants.kFortColor,
        borderRadius: BorderRadius.circular(35)),
      child: Center(
        child: Text(
          'Sign In',
          style: TextStyle(
            color: ColorConstants.kBlack,
            fontWeight: FontWeight.bold,
            fontSize: 18,
          ),
        ),
      ),
    ),
  ),
),
),
),
),
),
),
),
//register now
SizedBox(height: 15),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text(
      "Dont have an account? ",
      style: TextStyle(
        color: ColorConstants.kWhite,
        fontWeight: FontWeight.bold),
    ),
    GestureDetector(
      onTap: widget.showRegisterPage,
      child: Text(
        "Register Now!",
        style: TextStyle(
          color: ColorConstants.kFortColor,
          fontWeight: FontWeight.bold,

```

Listing 6 - login_screen.dart

U login_screen-u se nalazi stranica za prijavljivanje korisnika. Pomoću kontrolera se preuzimaju podaci iz polja: email i šifra, te se prosleđuju metodi za prijavljivanje. Ukoliko su uneti podaci korisnika ispravni, on dobija pristup aplikaciji.


```

class _TransactionsScreenState extends State<TransactionsScreen> {
  final AuthService service = AuthService();
  Future? currentAmount;
  Future? currentExpend;
  List transactionList = [];
  String selector = '';

  fetchDataBaseList() async {
    dynamic result = await service.getUserDataByCategory(selector);

    if (result == null) {
      debugPrint("cant get data");
    } else {
      setState(() {
        transactionList = result;
      });
    }
  }

  @override
  void initState() {
    currentAmount = service.getCurrentAmount();
    currentExpend = service.getCurrentExpend();
    fetchDataBaseList();
    super.initState();
  }

  body: Column(
    children: [
      Flexible(
        flex: 2,
        child: ListView.builder(
          shrinkWrap: true,
          itemCount: service.itemsExpend.length,
          scrollDirection: Axis.horizontal,
          itemBuilder: (context, index) => GestureDetector(
            onTap: () {
              setState(() {
                selector = service.itemsExpend.elementAt(index);
              });
            },
            child: Container(
              margin: EdgeInsets.all(5.0),
              alignment: Alignment.center,
              width: 100,
              height: 20,
              decoration: BoxDecoration(
                color: ColorConstants.kWhite,
                border: Border.all(
                  width: 1,
                ),
                borderRadius: BorderRadius.circular(15),
              ),
              child: Text(
                service.itemsExpend.elementAt(index),
                style: GoogleFonts.robotoMono(),
                textAlign: TextAlign.center,
              ),
            ),
          ),
        ),
      ),
    ],
  ),
)

```

```

    ),
    Expanded(
      flex: 20,
      child: FutureBuilder(
        future: service.getUserDataByCategory(selector),
        builder: (BuildContext context, AsyncSnapshot<dynamic>
snapshot) {
          return ListView.builder(
            itemCount: snapshot.data.length,
            itemBuilder: (context, index) {
              return Card(
                child: ListTile(
                  title: Text(
                    '${snapshot.data.elementAt(index) ['transactionName']}',
                    style: GoogleFonts.roboto(fontSize: 17),
                  ),
                  subtitle: Text(
                    '${snapshot.data.elementAt(index) ['transactionAmount']}',
                    style: GoogleFonts.roboto(fontSize: 15),
                  ),
                  trailing: Text(
                    '${snapshot.data.elementAt(index) ['category']}',
                    style: GoogleFonts.roboto(fontSize: 17),
                  ),
                ),
              );
            },
          );
        },
      ),
    ),
  ],
),

```

Listing 9- transactions_screen.dart isecak fajla

U isečku ovog fajla vide se delovi konstrukcije liste kategorija rashoda preko kojih se definiše identifikator koji će definisati transakcije koje spadaju u određenu kategoriju te ih izlistati na ekranu.

10. Zaključak

„*Aplikacija za menadžment budžeta*“ bi mogla da nađe svoju primenu u realnom svetu uz samo par dodataka. Kao prvi primer bih naveo davanje mogućnosti korisniku da manipuliše podacima svog profila, te bih tu dodao, da bi se u aplikaciju mogao integrisati bankovni račun. Na ovaj način bi se automatizovao unos iznosa transakcija.

Pored gore navedenih novina još bih dodao da bi se trebalo integrisati više analitike i da sama aplikacija podržava više valuta. Smatram da bi sa ovim izmenama „*Aplikacija za menadžment budžeta*“ mogla da pomogne mnogim ljudima u njihovim svakodnevnim životima.

11.Literatura

11.1 Knjige i zbornik

- [1] Mainkar P. i Giordano S. (2019). "Google Flutter Mobile Development Quick Start Guide". Packt
- [2] Merdžanović M. , Gašević D. , Havram M. , Hinić D. , Zbornik radova XLVII Konf za ETRAN, Herceg Novi, 8-13 juna 2003, tom III
- [3] Živković M. (2020). „Razvoj mobilnih aplikacija“. Univerzitet Singidunum

11.2 Internet stranice

- [1] Flater, <https://docs.flutter.dev/> [pristupljeno dana 03.10.2022. godine]
- [2] Firebase, <https://firebase.google.com/docs/build> [pristupljeno dana 04.10.2022. godine]
- [3] Dart, <https://dart.dev/guides> [pristupljeno dana 05.10.2022. godine]