Razvoj mobilne aplikacije za upravljanje osobnim financijama

Vuk, Antonio

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:939669

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-08**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANTONIO VUK

Razvoj mobilne aplikacije za upravljanje osobnim financijama

Diplomski rad

Pula, rujan 2021. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANTONIO VUK

Razvoj mobilne aplikacije za upravljanje osobnim financijama

Diplomski rad

JMBAG: 0303054766 redovni student
Studijski smjer: Informatika
Predmet: Napredni algoritmi i strukture podataka
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv.prof.dr.sc. Tihomir Orehovački

Pula, rujan 2021. godine

SAŽETAK

U ovome diplomskom radu izrađena je mobilna aplikacija za vođenje osobnih financija koristeći React Native. Aplikacija korisniku osim osnovnih funkcionalnosti omogućava i unos dodatnih polja za specifične potrebe, uvoz i izvoz podataka u Excel datoteku, način rada bez mreže, generiranje izvještaja i prikaz podataka pomoću grafikona. Izrađena aplikacije može se koristiti na android i iOS platformama. Aplikacija koristi REST API servis izrađen u Node.js i Express.js okruženju. Razvijeni servis služi za spremanje podataka u MySql bazu. Sve korištene tehnologije i funkcionalnosti aplikacije su detaljno opisane u radu i popraćene isječcima koda.

***KLJUČNE RIJEČI**: React Native, osobne financije, praćenje troškova, mobilna aplikacija*

ABSTRACT

In this thesis, a mobile application for managing personal finances using React Native was created. In addition to the basic functionalities, the application also enables the user to add additional custom fields for specific needs, import and export data to an Excel file, offline mode, generate reports and display data using charts. Developed application can be used on android and iOS platforms. The application uses the REST API service created in Node.js and Express.js environment. The developed service is used to save data to MySql database. All used technologies and functionalities of the application are described in detail and accompanied by code snippets.

***KEYWORDS**: React Native, personal finances, expense monitoring, mobile app*

SADRŽAJ

1. UVOD ... 1
2. POSTOJEĆE APLIKACIJE ZA PRAĆENJE TROŠKOVA ... 3
   2.1. ToshL Finance ... 4
   2.2. Spendee Budget & Money Tracker ... 5
   2.3. WalletApp by budgetbakery ... 6
3. VRSTE MOBILNIH APLIKACIJA ... 8
   3.1. Izvorne mobilne aplikacije ... 9
   3.2. Hibridne mobilne aplikacije ... 10
   3.3. Aplikacije za više platformi ... 11
4. KORIŠTENE TEHNOLOGIJE ... 12
   4.1. JavaScript ... 12
   4.2. React ... 14
   4.3. React Native ... 16
   4.4. Expo ... 17
   4.5. Node.js ... 18
   4.6. Express.js ... 19
   4.7. MySql ... 22
   4.8. SQLite ... 24
   4.9. Firebase ... 25
5. MOBILNA APLIKACIJA ZA PRAĆENJE OSOBNIH FINANCIJA ... 26
   5.1. Registracija i prijava korisnika ... 27
   5.2. Dodavanje, uređivanje i brisanje računa ... 29
   5.3. Dodavanje, uređivanje i brisanje kategorija ... 30
   5.4. Dodavanje, uređivanje i brisanje transakcija ... 31
   5.5. Kreiranje predložaka budućih transakcija ... 32
   5.6. Filtriranje transakcija ... 33
   5.7. Kreiranje obračuna ... 34
   5.8. Grafički prikaz troškova ... 36
   5.9. Kreiranje dodatnih polja ... 38
   5.10. Čitanje i pisanje podataka u Excel datoteku ... 39
   5.11. Način rada bez mreže ... 41
   5.12. Brisanje svih podataka ... 42
6. ZAKLJUČAK ... 43

LITERATURA ... 44

1. UVOD

Svatko od nas, na razne načine, svakodnevno koristi niz proizvoda i usluga koje je često potrebno platiti novcem. Za bolju kontrolu novca, lakše ostvarenje financijskih ciljeva i izbjegavanje prezaduženosti potrebno je pratiti troškove.

Za uspješno kreiranje i vođenje proračuna važno je razviti pozitivan stav prema upravljanju novcem.

Troškove je obavezno pratiti u firmama gdje dnevno cirkulira velika količina novca. Takve firme imaju posebne programe za kontrolu troškova i čitav tim koji se koristi tim programima. No praćenje troškova nije izrazito važno samo u firmama, već i na osobnoj razini.

U osobnim financijama važno je napraviti procjenu pridržavanja proračuna te izvršiti usporedbu tijekom vremena. Tako će se lakše uočiti eventualne slabosti i po potrebi ih korigirati.

Postoje različite metode kojima pojedinac može pratiti troškove. Neki se koriste rokovnicima, neki sakupljaju račune, neki koriste elektroničke programe kao što su Excel i Word. Ova rješenja pružaju pojedincu samostalno određivanje načina na koji će voditi troškove te koje će podatke bilježiti. Problem je što se dokumenti mogu izgubiti, a elektronički programi kao što su Excel i Word nisu uvijek dostupni.

Naglim razvojem tehnologije i sve većom primjenom pametnih uređaja javlja se potreba za modernijim metodama vođenja osobnih financija. Najpraktičnije rješenje je korištenje pametnih uređaja jer su uvijek dostupni.

Cilj ovoga rada je stvoriti mobilnu aplikaciju za vođenje osobnih financija i pri tome omogućiti korisnicima prilagođavanje aplikacije svojim osobnim potrebama. Postojeće aplikacije za vođenje financija omogućuje unos unaprijed definiranih polja, ako korisnik ima potrebu za unosom detaljnih podataka o nekoj kategoriji, korisnik tada nema rješenje, odustaje od korištenja aplikacije i vraća se staromodnim metodama vođenja financija koristeći Excel i Word alate. Aplikacija omogućuje korisnicima rad bez mreže, kreiranje izvještaja na temelju kojih korisnik može vidjeti razliku između prihoda i rashoda za odabrano razdoblje, filtriranje i grafički prikaz podataka.

Ovaj diplomski rad sastoji se od šest poglavlja, u drugom poglavlju napravljena je analiza tržišta, opisane su tri najkorištenije aplikacije za vođenje osobnih financija te je napravljena tablica usporedbe njihovih funkcionalnosti.

U trećem poglavlju opisane su vrste tehnologija za izradu mobilnih aplikacija, za potrebe ovog rada izrađena je aplikacija za više platformi.

Četvrto poglavlje opisuje korištene tehnologije u razvoju mobilne aplikacije, neke od tehnologija uključuju React Native za izradu mobilne aplikacije, Node.js i Express.js za komunikaciju između klijentske strane i MySql baze podataka.

U petom poglavlju predstavljen je konačan proizvod, njegov izgled, funkcionalnosti, objašnjen je slijed s kojim korisnik može uspješno koristiti aplikaciju.

U zaključku je razmotrena korisnost proizvoda te buduća poboljšanja.

Izvorni kod aplikacije dostupan je na GitHub-u:
Mobilna aplikacija: https://github.com/Antonio-Vuk/reactNative-financeApp
REST API: https://github.com/Antonio-Vuk/express-financeApp

2. POSTOJEĆE APLIKACIJE ZA PRAĆENJE TROŠKOVA

Svakodnevno pred svakim od nas stoje mnoge odluke. Većina odluka je jednostavna, poput onih „Što ću doručkovati?“ ili „Što ću danas obući?“, no postoje i one složenije „Koliko novca mogu potrošiti za kupnju hrane ili kupnju cipela?“.

Što je pojedinac uspješniji u donošenju odluka, život će mu biti kvalitetniji, uz manje briga, bolje će koristiti vrijeme, učinkovitije raspolagati novcem i naposljetku biti sretnija i zadovoljnija osoba.

Mudrost upravljanja novcem bitna je za pojedinca, ali i za cijelu obitelj. Važno je naučiti upravljati novcem, kako novac ne bi upravljao nama. Nažalost, mnogo ljudi ne zna upravljati vlastitim financijama. Za dobro i uspješno upravljanje novcem, potreban je jasan i konkretan financijski plan kako bi se ostvario cilj na najbolji mogući način.

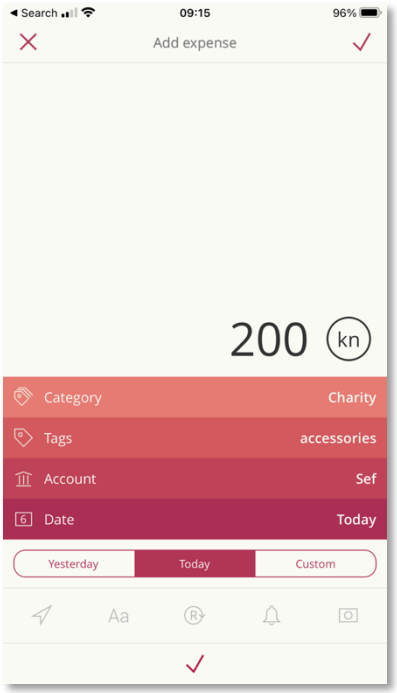
Prije nego se krene u proces financijskog planiranja, važno je znati procijeniti trenutno financijsko stanje, odnosno utvrditi financijsku sposobnost za ostvarenje nekog cilja. To je važno zbog uvida u količinu novca na raspolaganju, koliko ga se troši, od kuda dolazi te koliko ga na kraju ostaje.

Kao što se more diže i spušta za vrijeme plime i oseke, tako se kreće i novac, s time da se može kontrolirati kako novac dolazi i odlazi kroz proračun kućanstva (Barbić, n. d.).

U nastavku je dan pregled postojećih aplikacija za praćenje troškova i na kraju je napravljena tablica usporedbe njihovih funkcionalnosti.

2.1. ToshL Finance

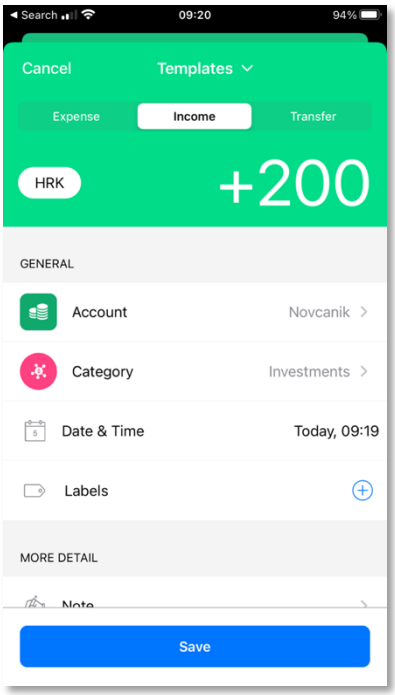
ToshL je mobilna aplikacija za praćenje financija koja korisnicima omogućava praćenje i upravljanje dnevnim troškovima, kao i organiziranje računa. Dostupna je na iOS, windows i android platformama. Aplikacija omogućava izradu grafikona, korištenje otiska prsta za otključavanje, unos kategorija, sustave označavanja, konverzije valuta u stvarnom vremenu i još mnogo toga (Toshl, n. d.). Na slici 1. prikazan je ekran za unos transakcija iz aplikacije ToshL Finance.



Slika 1. Prikaz ekrana za unos transakcija iz aplikacije ToshL Finance

2.2. Spendee Budget & Money Tracker

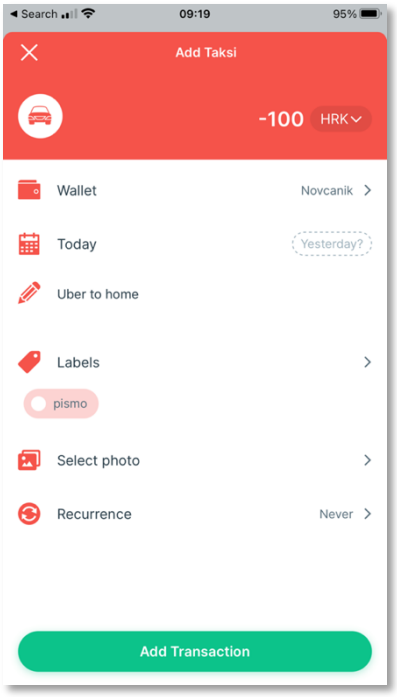
Spendee Budget je aplikacija dostupna za iOS platformu. Besplatna je i nudi mogućnost kupnje unutar aplikacije. Trenutno ima više od tri milijuna korisnika širom svijeta. Ova aplikacija omogućava stvaranje proračuna, kreiranje novčanika (eng. wallet), dijeljenje financija, više valuta, oznake, noćni način rada, sigurno sinkroniziranje podataka, a dostupna je i web verzija. Korištenje aplikacije je jednostavno zahvaljujući dizajnu. Spendee tim ulaže velike napore u dizajn aplikacije za koji je dobio i nagrade (Spendee, n. d.). Na slici 2. prikazan je ekran za unos transakcija iz aplikacije Spendee.



Slika 2. Prikaz ekrana za unos transakcija iz aplikacije Spendee

2.3. WalletApp by budgetbakery

BudgetBakers je projekt osnovan 2010. godine koji razvija alat za upravljanje osobnim financijama za osobne potrebe i potrebe malih poduzeća. Ova aplikacija je dostupna na iOS i android platformi, a omogućeno je korištenje i preko web-a. Trenutno ima više od pet milijuna korisnika i četrnaest članova u razvojnom timu. Aplikacija omogućava ručni unos podataka, unos podataka iz CSV datoteke, stvaranje novčanika te korištenje različitih valuta po novčaniku. Većina mogućnosti je dostupna besplatno, a za neke je potrebno platiti (Budgetbakers, n. d.). Na slici 3. prikazan je ekran za unos transakcija iz aplikacije WalletApp.



Slika 3. Prikaz ekrana za unos transakcija iz aplikacije WalletApp

U tablici 1. napravljena je usporedba prethodno navedenih aplikacija u pogledu osnovnih funkcionalnosti. Za svaku od funkcionalnosti navedeno je, sadržava li ju pojedina aplikacija te dolazi li u besplatnoj ili plaćenoj verziji.

	WalletApp by budgetbakery	**Plaćanje**	**ToshL Finance**	**Plaćanje**	**Spendee Budget & Money Tracker**	**Plaćanje**
Mjesečni plan	30 kn	-	22 kn	-	25 kn	-
Godišnji plan	175 kn	-	150 kn	-	190 kn	-
Neograničeni plan	379 kn	-	-	-	-	-
Besplatni period	DA	NE	DA	NE	DA	NE
Neograničeni računi	NE	DA	NE	DA	NE	DA
Dodavanje kategorija	NE	NE	DA	NE	DA	NE
Rad bez interneta	DA	NE	DA	NE	DA	NE
Importiranje datoteka	DA	NE	NE	NE	NE	NE
Podsjetnici	NE	NE	NE	DA	NE	NE
Otključavanje otiskom prsta	DA	NE	NE	DA	NE	NE
Dijeljenje troškova s grupama	DA	NE	NE	NE	NE	DA
Spremanje fotografija	DA	NE	NE	NE	DA	DA

Tablica 1. Usporedba funkcionalnosti aplikacija

3. VRSTE MOBILNIH APLIKACIJA

Danas je gotovo nemoguće provesti dan ne koristeći pametni uređaj. Od početka dana kada se ugasi alarm, pa tijekom cijelog dana dok se gledaju YouTube videa, prate novosti na mrežnim portalima, lista po društvenim mrežama, pametni uređaji postaju neizostavan dio svakodnevice.

Koriste se stotine novih aplikacija svake godine te potražnja za mobilnim aplikacijama nastavlja rasti. Poznati su operativni sustavi kao što su android i iOS, no ipak nisu toliko poznate specifične tehnologije koje programeri koriste za izradu mobilnih aplikacija.

Većina poduzeća uz svoje web aplikacije ima potrebu za mobilnim verzijama, no prelazak na mobilno poslovanje nije jednostavno. Većina se kompanija susreće s problemom odabira prave tehnologije.

Zbog ograničenja koje pružaju mobilni uređaji kao što su veličina ekrana, manje procesorske i grafičke mogućnosti u odnosu na desktop računala, neke kompanije ne mogu prenijeti kompletno poslovanje na mobilne uređaje. Zbog tolike dostupnosti mobilnih uređaja, takve kompanije pokušavaju prenijeti dio poslovanja ili samo neke od funkcionalnosti kao što su unos i uređivanje podataka, a vizualizaciju podataka, generiranje izvještaja i ostale funkcionalnosti za koje je potreban veliki ekran, ostavljaju za svoje web i desktop aplikacije.

U nastavku su objašnjene tehnologije izrade mobilnih aplikacija te za svaku su navedeni prednosti i nedostatci.

3.1. Izvorne mobilne aplikacije

Aplikacije ovog tipa specifično su dizajnirane za jedan operacijski sustav, kako bi se to omogućilo, tu su određene tehnologije i programski jezici koje programeri koriste. Na primjer, ako se izrađuje aplikacija za Appleov iOS koristi se Objective-C ili Swift, dok za Googleov android programeri koriste Javu ili Kotlin. Za razvoj izvornih mobilnih aplikacija i android i iOS programeri koriste specifičan skup razvojnih alata (eng. Software Development Kit – SDK) te integrirano razvojno okruženje (eng. Integrated Development Environment - IDE). SDK pruža alate za programiranje, dok IDE je samo korisničko sućelje koje se sastoji od komponenti potrebnih za programiranje (ETraverse, n. d.).

Prednosti:

- **bolja funkcionalnost**: nema ograničenja pristupu sučeljima i alatima na platformi za koju se razvija.
- **veće performanse i odlično korisničko iskustvo**: izravna interakcija između programskog koda i temeljnih resursa rezultira visokim performansama. Također, izvorne aplikacije općenito pružaju bolje korisničko iskustvo.

Nedostatci:

- **trošak izrade:** izrada izvornih aplikacija je skupa kada se razvija aplikacija za više platformi. To znači da je potrebno zaposliti dva razvojna tima koja će raditi na različitim platformama.
- **vrijeme izrade:** razvoj izvornih aplikacija zahtjeva puno vremena jer se rad obavljen za jednu platformu ne može duplicirati na drugu. Umjesto toga, potreban je poseban tim za rad na drugoj platformi.

3.2. Hibridne mobilne aplikacije

Ovaj tip aplikacija instalira se na uređaj baš kao i izvorne mobilne aplikacije. Sve hibridne aplikacije razvijene su HTML i CSS programskim jezicima, pokreću se pomoću pojednostavljenog web preglednika unutar aplikacije. U situaciji kada postoji ideja i cilj u što kraćem roku staviti korisnicima aplikaciju na raspolaganje, a pritom su ograničeni resursi, ovaj tip aplikacija nudi rješenje (Saccomani, n. d.).

Prednosti:

- **trošak izrade:** hibridne aplikacije lakše je prebaciti na drugu platformu. Jednom kada se izradi aplikacija za jednu platformu s lakoćom se pokreće na drugoj.
- **vrijeme izrade:** budući da je potrebno razvijati samo jedan programski kod, potrebno je duplo manje programera nego kod izvornih mobilnih aplikacija. Ili bi se, s istim brojem programera hibridna aplikacija mogla proizvesti u pola vremena.

Nedostatci:

- **performanse:** budući da se sadržaj aplikacije prikazuje kroz pojednostavljeni web preglednik, performanse još nisu dosegle razinu kao kod izvornih aplikacija.
- **korisničko iskustvo:** korisnici iOS i androida vrlo su lojalni svojim platformama, a budući da ih koriste već godinama, navikli su na to kako stvari funkcioniraju u izvornim aplikacijama.

3.3. Aplikacije za više platformi

Aplikacije za više platformi ne mogu se mjeriti s izvornima kada su u pitanju performanse. No mogućnost pokretanja jedne baze koda na nekoliko različitih operativnih sustava znatno utječe na odluku o odabiru tehnologije za razvoj mobilne aplikacije. Mobilni operacijski sustavi imaju svoj izvorni SDK koji se koristi za njihov razvoj. Srećom, ti SDK-ovi također podržavaju aplikacijska programska sučelja koja se koriste u programskim jezicima različitim od izvorno definiranog. Razvojni alat za izradu aplikacija za više platformi stvaraju različiti dobavljači koji koriste SDK izvornih operacijskih sustava.

Prednosti:

- **trošak i vrijeme izrade:** umjesto dva tima programera potreban je samo jedan tim koji razvija programski kod za više platformi.
- **korisničko iskustvo:** s obzirom na to da SDK aplikacije je izgrađen nad SKD-om izvornih mobilnih aplikacija, komponente su gotovo identične, a razlike su minimalne.

Nedostatci:

- **performanse:** kada su u pitanju grafički i procesorski teški zadatci, postoji značajna razlika između izvornih i aplikacija za više platformi.
- **ograničeni resursi:** nisu sve značajke i funkcionalnosti izvornih aplikacija dostupni kada su u pitanju aplikacije za više platformi. To potiče programere da razmotre alternative ili pronađu način za integraciju željene funkcionalnosti u mobilnu aplikaciju, što oduzima puno vremena.

Postoje mnoge kompanije koje nude rješenja za razvoj aplikacija za više platformi. Neki od najpoznatijih su Flutter, React Native i Xamarin. Google je počeo razvijati Flutter 2015. godine, no svoju beta verziju je dostigao tek 2018. godine. Od tada ga programeri koriste za razvoj aplikacija za više platformi koje djeluju gotovo jednako kao izvorne. Xamarin je vrhunac popularnosti dostigao 2019. godine, no danas polako gubi bitku. Xamarin koristi C# programski jezik (Luetić, 2021).

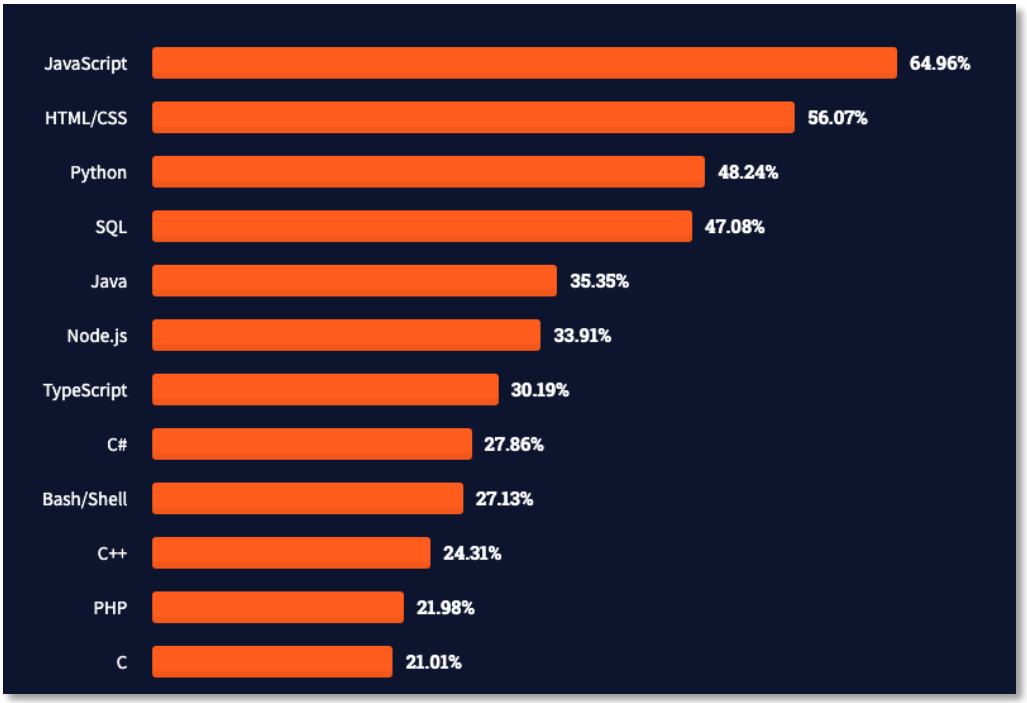
4. KORIŠTENE TEHNOLOGIJE

Cijeli sustav je strukturiran na više manjih podsustava koji međusobno komuniciraju i razmjenjuju podatke. Sustav ove aplikacije dizajniran je koristeći arhitekturu klijent-poslužitelj. U ovoj aplikaciji poslužitelj je Node.js server, a klijent je mobilna aplikacija. Komunikacija između klijenta i poslužitelja odvija se koristeći HTTP protokol. Podaci se prenose koristeći JSON (eng. JavaScript Object Notation).

4.1. JavaScript

JavaScript je jedan od najpoznatijih i najkorištenijih programskih jezika. Raste puno brže od bilo kojeg drugog programskog jezika te velike svjetske kompanije kao što su PayPal, Netflix, Walmart, Uber i drugi koriste JavaScript za izradu svojih aplikacija.

Na slici 4. prikazan je popis najpopularnijih tehnologija u 2021. godini. JavaScript je devet godina zaredom najčešće korišten programski jezik (Stackoverflow, 2021).



Slika 4. Prikaz najkorištenijih tehnologija u 2021. godini

JavaScript je jednostavan, interpretiran programski jezik namijenjen ponajprije

razvoju interaktivnih HTML-stranica. Jezgra JavaScripta uključena je u većinu današnjih preglednika. Firefox koristi SpiderMonkey, Chrome koristi V8 te zbog tolike raznolikosti prevoditelja ponekad se JavaScript ponaša drugačije na različitim preglednicima.

JavaScript nije pojednostavljena inačica programskog jezika Java. Povezuje ih jedino slična sintaksa i izvršavanje određenih radnji unutar preglednika. Izvorno JavaScript se trebao zvati LiveScript, ali da bi se potakla uporaba novog skriptnog jezika, nazvan je slično jeziku Java, od kojeg se u tadašnje vrijeme dosta očekivalo (Crockford, 2008).

JavaScript se razvija od 1995. godine kada je Netscape objavio nekoliko prvih inačica jezika. Nedugo nakon toga Microsoft je objavio jezik sličan JavaScriptu pod nazivom Jscript. Danas je za standardizaciju skriptnih jezika, pa tako i JavaScripta, zadužena organizacija ECMA. Standardi se objavljuju pod nazivom ECMAScript i do sada je objavljeno pet inačica standarda ECMA-262 (Stančer, 2015).

Na slici 5. prikazana je metoda za izračunavanje stanja računa. Navedena metoda je napisana u JavaScript programskom jeziku. Za izračunavanje stanja na računu potrebno je proći kroz sve transakcije te provjeriti tip transakcije, pripada li transakcija odabranom računu, te ovisno o tipu transakcije ažurirati stanje računa.

```
const getWalletBallance = (wallet) => {
    let initialBallance = wallet.ballance;
    defaultState.transactions.forEach((transaction) => {
        if (transaction.type == constants.income) {
            if (transaction.toAccountId == wallet.id) {
                initialBallance = +initialBallance + +transaction.amount;
            }
        }
        if (transaction.type == constants.expense) {
            if (transaction.fromAccountId == wallet.id) {
                initialBallance = +initialBallance - +transaction.amount;
            }
        }
        if (transaction.type == constants.transfer) {
            if (transaction.fromAccountId == wallet.id) {
                initialBallance = +initialBallance - +transaction.amount;
            }
            if (transaction.toAccountId == wallet.id) {
                initialBallance = +initialBallance + +transaction.amount;
            }
        }
    });
    return initialBallance;
};
```

Slika 5. Programski kod za izračunavanje stanja novčanika

4.2. React

React je JavaScript biblioteka za izradu korisničkih sučelja. Razvijen je od strane Facebooka 2011. godine i trenutno je jedna od najpoznatijih JavaScript biblioteka za izradu korisničkog sučelja uz Angular i Vue.js. Kada se izrađuje aplikacija koristeći React, ustvari se izrađuju neovisne i više puta iskoristive komponente od kojih se sastavljaju kompleksna korisnička sučelja. Svaka React aplikacija sastoji se od barem jedne komponente koja se naziva glavna komponenta, ta komponenta sadrži druge komponente te tako se stvara cijelo stablo komponenti (Maiti i Bidinger, 1981). Na slici 6. prikazana je glavna komponenta izrađene aplikacije.

```
const App = () => {
    const [state, setState] = useState(defaultState);

    const [loaded] = useFonts({
        "Roboto-Black": require("./app/assets/fonts/Roboto-Black.ttf"),
        "Roboto-Regular": require("./app/assets/fonts/Roboto-Regular.ttf"),
        "Roboto-Bold": require("./app/assets/fonts/Roboto-Bold.ttf"),
    });
    useEffect(() => {
        LogBox.ignoreLogs([
            "VirtualizedLists should never be nested inside plain ScrollViews",
            "with the same orientation - use another VirtualizedList-backed container instead.",
            "Non-serializable values were found in the navigation state.",
        ]);
        createTables();
    }, []);

    useEffect(() => {
        loadData(setState);
    }, []);

    if (!loaded) {
        return null;
    }
    return (
        <>
            <ActivityIndicatorComponent visible={state.loading} />
            {loaded && (
                <AppContext.Provider value={{ state, setState }}>
                    <NavigationContainer>
                        {state.onBoard != "true" && <OnBoardingScreen />}
                        {state.onBoard == "true" && (
                            <>
                                <FlashMessage position="top" />
                                {state.user != undefined && <AppNavigator />}
                                {state.user == undefined && <AuthNavigator />}
                            </>
                        )}
                    </NavigationContainer>
                </AppContext.Provider>
            )}
        </>
    );
};
```

Slika 6. Glavna komponenta aplikacije

Što se tiče implementacije, komponenta je implementirana kao JavaScript klasa koja ima svoje stanje i metodu za renderiranje. Stanje predstavljaju podaci koji se prikazuju kada se komponenta renderira, a metoda je odgovorna za izgled korisničkog sučelja. Output render metode je React element, on je zapravo jednostavan JavaScript objekt, koji se prenosi na objektni model dokumenta (eng. Document Object Model – DOM). React sadrži prikaz DOM-a u memoriji koji se naziva virtualni DOM. Za razliku od DOM-a u pregledniku, ovaj DOM je lagan za kreiranje. Promjenom stanja komponente dobiva se novi React element. React uspoređuje taj element s djecom prethodnog, shvaća što se promijenilo i ažurira pravi DOM kako bi bio sinkroniziran s virtualnim. To znači kada se izrađuje aplikacija koristeći React, nije potrebno raditi s DOM sučeljem u pregledniku. Odnosno ne piše se kod koji će manipulirati DOM-om te dodavati evente DOM elementima. Jednostavno se promijeni stanje komponente i React automatski izmijeni izgled te komponente. Zato se ova biblioteka zove React, jednostavno reagira na promjene stanja komponenti (Maiti i Bidinger, 1981).

Jedno od pitanja koje se često postavlja je React ili Angular, oni su slični što se tiče arhitekture sastavljene od komponenti. Angular je potpuno rješenje, no kod rada s React-om potrebno je koristiti druge biblioteke za pozivanje HTTP zahtjeva ili drugih funkcionalnosti, što nije loše jer pruža slobodu programerima u odabiru načina na koji žele odrađivati stvari, dok Angular ne pruža tu slobodu.

4.3. React Native

Facebook je kreirao React Native za izradu svojih mobilnih aplikacija. Motivacija je došla iz činjenice što je React za web bio jako uspješan. Ako je React tako dobar alat za izradu korisničkog sučelja, a postoji potreba za izvornim mobilnim aplikacijama, bilo je potrebno samo omogućiti React-u rad s komponentama izvornih mobilnih aplikacija. Kako postoji velika potražnja za mobilnim aplikacijama, a puno programera je znalo React, bilo im je jednostavno prilagoditi se i naučiti novu tehnologiju za izradu mobilnih aplikacija (Eisenman, 2016).

Mnoge velike kompanije koriste React Native za izradu svojih mobilnih aplikacija, neke od najpoznatijih su Instagram, Pinterest, Skype, Uber te mnogi drugi.

Na slici 7. prikazan je programski kod React Native komponente. Prikazana komponenta sastoji se samo od metode za renderiranje pošto služi za prikaz tipke. Kao parametar prima funkciju po nazivu „onPress“, navedena funkcija se izvodi pozivanjem „onPress“ eventa „ThouchableOpacity“ komponente iz React Native biblioteke.

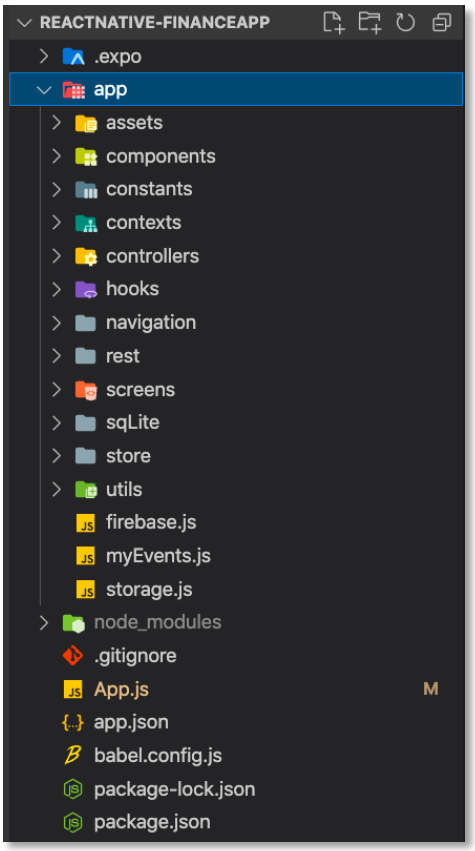
```
import React from "react";  7.4K (gzipped: 3K)
import { TouchableOpacity } from "react-native";
import { FontAwesome } from "@expo/vector-icons";
import { COLORS, icons, SIZES } from "../constants";

const CircleButton = ({ onPress }) => {
    return (
        <TouchableOpacity
            onPress={onPress}
            style={{
                width: 50,
                justifyContent: "center",
                alignItems: "center",
                backgroundColor: COLORS.primary,
                borderRadius: 25,
            }}
        >
            <FontAwesome
                name={icons.check}
                size={SIZES.icon}
                color={COLORS.white}
            />
        </TouchableOpacity>
    );
};

export default CircleButton;
```

Slika 7. Programski kod React Native komponente

4.4. Expo

Postoje dva načina izrade React Native aplikacija. Moguće je koristiti čisti React Native ili Expo. Expo je skupina alata izrađena koristeći React Native koja dodatno olakšava proces izrade mobilnih aplikacija. Expo je odličan za programere koji tek kreću s izradom mobilnih aplikacija te nemaju iskustva s izradom izvornih mobilnih aplikacija.

Kada se kreira projekt koristeći React Native, projekt se sastoji od iOS i android projekata, sadržava i dodatni kod koji se dijeli između navedenih projekata. Projekt kreiran koristeći Expo ne sadržava iOS i android projekte već samo jedan projekt koji koristi JavaScript programski jezik. Struktura takvog projekta prikazana je na slici 8. Zbog toga nije moguće raditi direktno s izvornim sučeljem ovih platformi, već samo s onim što pruža Expo po pitanju izvornih mogućnosti. Bez obzira na navedeno ograničenje, moguće je izraditi potpunu aplikaciju koristeći samo Expo. Ako se u projektu pokaže potreba za nekom od mogućnosti iOS i android platformi, uvijek postoji mogućnost prebacivanja iz Expo okruženja u React Native.



Slika 8. Struktura izrađenog projekta koristeći Expo

4.5. Node.js

Node.js je okruženje za izvršavanje JavaScript koda. JavaScript se koristio za izradu aplikacija koje se pokreću samo unutar web preglednika što danas više nije slučaj. Ryan Dahl je 2009. godine uzeo JavaScript Google V8 prevoditelj, ugradio ga u C++ program i nazvao ga Node.js. On se također sastoji od dodatnih modula koji nisu dostupni unutar web preglednika.

Pomoću Node.js-a se izrađuju servisi i sučelja za aplikacije (eng. Application programming interface – API). To su usluge koje koriste klijentske aplikacije, bilo to desktop, web ili mobilna aplikacija. Te klijentske aplikacije su samo površina koju korisnik vidi i s čime komunicira. No one moraju komunicirati s uslugama koje se nalaze na serveru, kako bi pohranili podatke, slali elektroničku poštu te obavljali ostale zadatke. Node.js je prepoznat i od velikih kompanija kao što su PayPal, Google, Netflix i drugi. Jedna od prednosti Node.js-a je što koristi JavaScript, tako da programeri koristiti svoje JavaScript vještine za izradu servisa i sučelja za klijentske aplikacije. Node.js ima veliku ponudu biblioteka otvorenog koda, za svaki zahtjev postoji besplatna biblioteka koja se može koristiti. Bitno je napomenuti da je Node.js asinkrone prirode, što znači da koristi jednu jezgu za posluživanje više zahtjeva (Brown, 2016).

4.6. Express.js

Express.js je minimalna i fleksibilna biblioteka koja pruža robustan skup alata za mobilne i web aplikacije. Uz hrpu pomoćnih metoda i alata na raspolaganju, stvaranje robusnog API-ja je brzo i jednostavno (Brown, 2016).

Pokretanje servera pomoću express.js biblioteke je jednostavno i lagano, to je moguće uz samo par linija koda prikazanih na slici 9.

```
index.js > ...
1   const express = require("express");
2   const app = express();
3   app.use(express.json());
4
5   require("./startup/routes")(app);
6
7   const port = 3000;
8   app.listen(port, () => console.log("Backend se vrti na portu: " + port));
9
```

Slika 9. Programski kod za pokretanje servera koristeći express.js biblioteku

Kao što je jednostavno i lako pokrenuti server tako je jednostavno kreirati i akcije koje će odgovarati na zahtjeve klijenata. Slika 10. prikazuje programski kod akcije za kreiranje transakcije.

```
router.post("/", auth, async (req, res) => {
    try {
        const { error } = validateTransaction(req.body.transaction);
        if (error) {
            return res.status(400).send(error.details[0].message);
        }

        const transaction = await insertTransaction(
            req.body.transaction,
            req.user.id
        );

        const customFieldValues = await insertCustomFieldValues(
            transaction.id,
            req.body.customFields
        );

        return res.send({ transaction, customFieldValues });
    } catch (error) {
        return res.status(405).send(error.message);
    }
});
```

Slika 10. POST akcija za unos transakcija

Za unos transakcije korištena je POST metoda. Prikazana metoda ima tri parametra, prvi je url akcije, drugi parametar je posrednik za autentifikaciju u kojem je korišten JWT (eng. Jason Web Token). Programski kod posrednika prikazan na slici 11. treći parametar je funkcija koja se izvršava.

```
const jwt = require("jsonwebtoken");
const auth = (req, res, next) => {
    const token = req.header("x-auth-token");
    if (!token) {
        return res.status(401).send("Access denied. No token provided");
    }

    try {
        const decoded = jwt.verify(token, "jwtPrivateKey");
        req.user = decoded;
        next();
    } catch (ex) {
        return res.status(400).send("Invalid token.");
    }
};

module.exports = auth;
```

Slika 11. Posrednik za autentifikaciju

Na početku metode provjerava se pravilnost tijela zahtjeva. Za validaciju je korištena biblioteka Joi. Nakon validacije transakcija se unosi u MySql bazu, a zatim se spremaju i dodatna polja. Na kraju se korisniku vraća odgovor koji sadrži podatke unesene u bazu. Podaci koji se prenose u JSON formatu prikazani su na slici 12. S desne strane prikazan je zahtjev, a s lijeve strane prikazan je odgovor.

Slika 12. Zahtjev i odgovor u JSON formatu POST akcije za unos transakcije.

U tablici 2. nalazi se popis svih metoda u izrađenom API servisu.

HTTP Akcija	URL	OPIS
POST	api/auth	registracija korisnika
POST	api/users	prijava korisnika
POST	api/wallet	kreiranje računa
PUT	api/wallet	uređivanje računa
DELETE	api/wallet/:id	brisanje računa
POST	api/category	kreiranje kategorije
PUT	api/category	uređivanje kategorije
DELETE	api/category/:id	brisanje kategorije
GET	api/data	učitavanje svih podataka
DELETE	api/data	brisanje svih podataka
POST	api/transaction	kreiranje procesirane transakcije
POST	api/transaction/incoming	kreiranje transakcije za template
PUT	api/transaction/:id	uređivanje transakcije
DELETE	api/transaction/:id	brisanje transakcije
POST	api/customField	kreiranje polja
PUT	api/customField	uređivanje polja
DELETE	api/customField/:id	brisanje polja
POST	api/customFieldListValue	kreiranje naziva elementa liste
PUT	api/customFieldListValue	uređivanje naziva elementa liste
DELETE	api/customFieldListValue/:id	brisanje naziva elementa liste

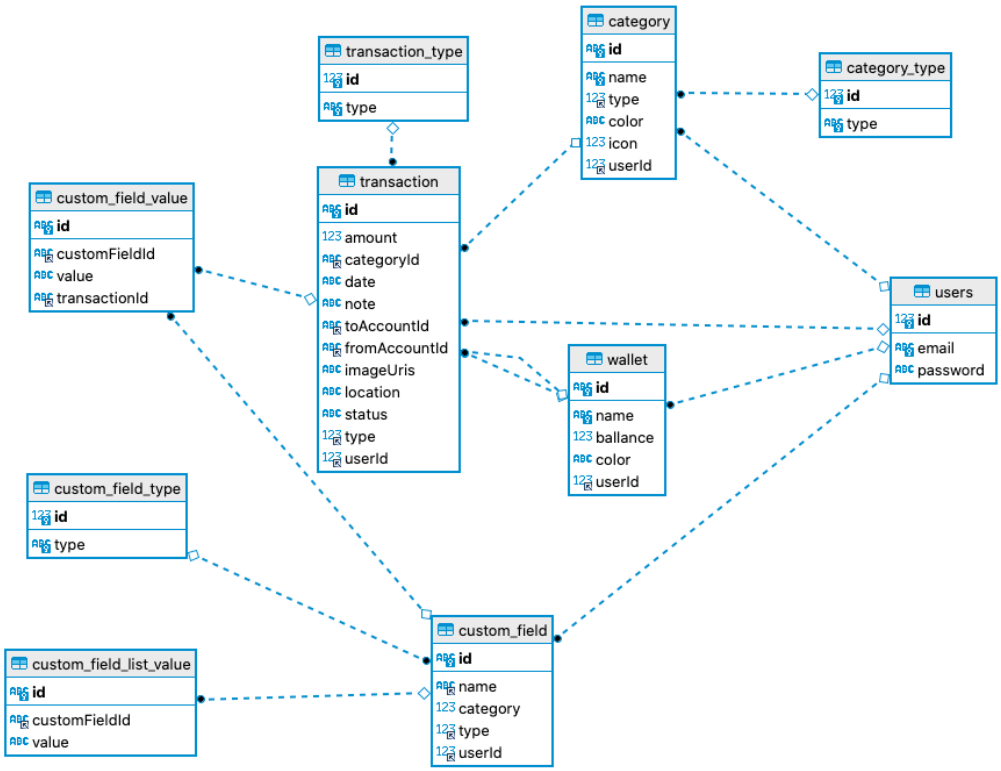
Tablica 2. Popis svih metoda REST API-a

4.7. MySql

Najvažniji dio svake aplikacije su podaci. Za očuvanje podataka važno ih je pohraniti u bazu podataka. Za rad s bazom podataka koristi se sustav za upravljanje bazom podataka (eng. Database Management System – DBMS). Poveže se s DBMS-om i šalju se upiti za dohvaćanje i uređivanje podataka. DBMS izvršava instrukcije i vraća rezultat. Postoje različiti sustavi za upravljanje podacima, no svi oni su podijeljeni u dvije kategorije: relacijske i NoSQL. U relacijskom bazama pohranjuju se podaci u tablice te se povezuju koristeći veze. Za upravljanje takvim sustavima koristi se SQL (eng. Structured Query Language) jezik. MySql je jedna od najpoznatijih baza podataka otvorenog koda na svijetu.

MySql poslužitelj namijenjen je i za kritične te visoko opterećene proizvodne sustave kao i za softver za masovnu primjenu. MySql je proizvela tvrtka Oracle, dostupan je pod slobodnom licencom, ali moguća je i kupnja standardne komercijalne licence (Manual, 2013).

Struktura baze podataka korištene za izradu aplikacije prikazana je na slici 13.



Slika 13. Struktura baze podataka

U tablicu „users“ spremaju se podaci o korisnicima koji su registrirani u aplikaciju. Navedena tablica sadržava e-poštu korisnika i kriptiranu zaporku.

U tablicu „wallet“ spremaju su podaci o računima, svaki račun sastoji se od naziva, stanja, boje i podatka o korisniku koji je kreirao račun.

U tablicu „category_type“ sprema se podatak o tipu kategorije, on može biti prihod, rashod ili oboje.

U tablicu „category“ sprema se podatak o korisniku kojem pripada pojedina kategorija, podatak o tipu kategorije, naziv, boja i ikona.

U tablicu „transaction_type“ sprema se podatak o tipu transakcije koji može biti prihod, rashod ili transfer. Transfer služi za prebacivanje sredstva s jednog računa na drugi.

Tablica „transaction“ je glavna tablica ove aplikacije u koju se spremaju zapisi o potrošnji korisnika. Tablica sadrži vrijeme stvaranja transakcije, iznos transakcije, bilješku korisnika o transakciji, zapis o korisniku koji je dodao redak, podatak o kategoriji kojoj pripada transakcija, tip transakcije, ako je transakcija prihod onda se sprema id računa na koji novac dolazi „toAccountId“, ako je transakcija rashod onda se sprema id računa s kojeg novac odlazi „fromAccountId“. Ako je transakcija tipa transfer onda se sprema id računa u polje „toAccountId“ i „fromAccountId“. Ova tablica također sadrži podatak o slikama koje korisnik učita s uređaja, lokaciju na kojoj je transakcija učitana, status transakcije, on može biti „Processed“ ili „Pending“. Ako je na čekanju „Pending“ to znači da se taj zapis koristi kao uzorak za buduće unose.

U tablicu „custom_field“ spremaju se podaci o dodatnim poljima korisnika. Za svaku transakciju korisnik bilježi dodatne podatke ovisno o potrebama. Za svako polje određuje se naziv, tip i kategorija kojoj pripada. Može pripadati i svim kategorijama, onda se unosi numerička vrijednost 0.

U tablicu „custom_field_type“ spremaju su podaci o tipu „custom polja“, oni mogu biti lista, broj, datum ili tekst.

U tablicu „custom_field_list_value“ spremaju se elementi „custom polja“ ako je tip lista. Korisnik unosi elemente koje može odabrati iz izbornika.

4.8. SQLite

SQLite je programski paket u javnoj domeni koji pruža sustav upravljanja relacijskim bazama podataka. SQLite je lagan što se tiče složenosti postavljanja i trošenja resursa. Ne zahtijeva zaseban poslužiteljski proces ili sustav za rad, već izravno pristupa datotekama za pohranu. Pošto nema poslužitelja, to znači da nema dodatnog postavljanja. Stvaranje instance SQLite baze podataka jednostavno je i lako kao i otvaranje datoteke. Cijela baza se nalazi u jednoj datoteci koja se može koristiti na više platformi. Osnovna verzija baze ima manje od megabajt i zahtjeva samo par megabajta memorije za rad (Using SQLite, n. d.). Zbog navedenih karakteristika ova baza je neizostavna za omogućavanje rada aplikacije bez mreže.

Struktura korištene SQLite baze u ovome radu je slična strukturi MySql baze korištene za izradu aplikacije. Jedina razlika je što nema tablicu „users“, pošto način rada bez mreže je omogućen samo za jednog korisnika „vlasnika mobilnog uređaja“. Kod rada bez mreže nije moguće učitati slike na Firebase server, iz tog razloga prilikom spremanja transakcija u SQLite bazu izostavljeni su podaci o slikama.

Na slici 14. prikazana je pomoćna metoda za rad s SQLite bazom podataka. Navedena metoda prima tri parametra. Prvi parametar je metoda koja se izvršava nakon završenog upita. Drugi parametar je sql naredba, treći parametar je polje argumenata čiji se elementi ubacuju u sql upit. Za izvršenje upita potrebno je izvršiti sql naredbu koristeći transakciju iz SQLite biblioteke.

```
import * as SQLite from "expo-sqlite";  10K (gzipped: 3.7K)
const db = SQLite.openDatabase("myDatabase");
import { sql } from "./sqLiteSql";

const queryDatabase = (myFunction, slqStatement, argsArray = []) => {
    db.transaction((tx) => {
        tx.executeSql(
            slqStatement,
            argsArray,
            (tx, data) => {
                myFunction({ success: true, data: data });
            },
            (tx, error) => {
                myFunction({ success: false, data: error.message });
            }
        );
    });
};
```

Slika 14. Pomoćna metoda za rad sa SQLite bazom podataka

4.9. Firebase

Firebase spremnik u oblaku namijenjen je programerima aplikacija koji trebaju pohranjivati i posluživati sadržaj koji stvaraju korisnici, poput fotografija ili videozapisa. Operacije preuzimanja i slanja podataka su vrlo učinkovite bez obzira na kvalitetu mreže, što znači da prilikom nestanka mreže, te njenog povratka ponovno pokreću tamo gdje su stale, štedeći korisnicima vrijeme. Firebase je izgrađen za vrlo zahtjevne korisnike, veličina spremnika se prilagođava potrebama, stoga nije potrebno mijenjati pružatelja usluge kada se obujam podataka poveća (Firebase, n. d.).

Na slici 15. prikazan je programski kod potreban za učitavanje slika na Firebase. Prvo se poziva metoda „uploadImage“ koja za parametre prima url slike i indeks koji se koristi za imenovanje slike na serveru. Navedena metoda se koristi u metodi „uploadImagesToFirebase“ koja kao parametar prima polje slika.

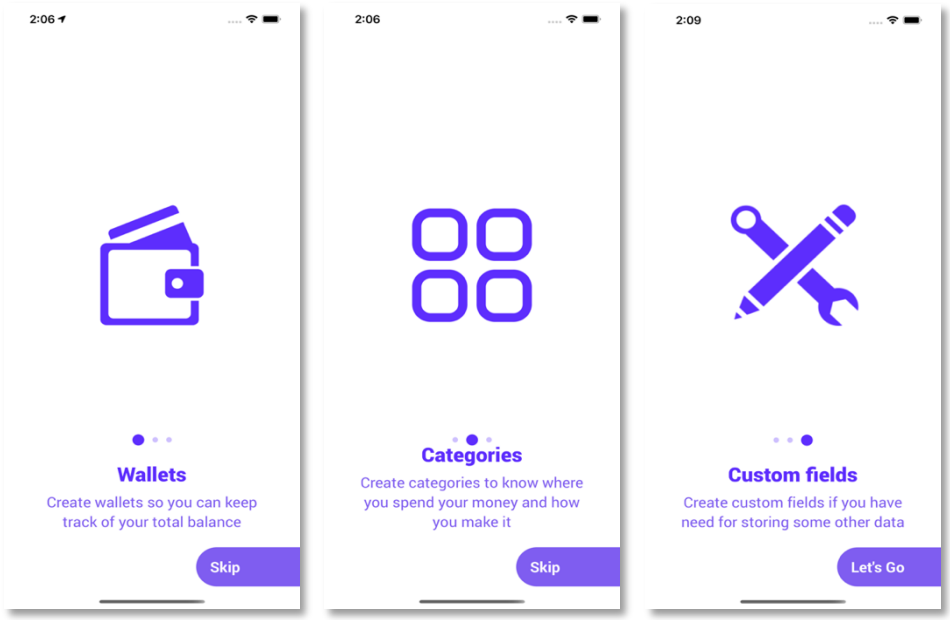
```
import * as firebase from "firebase";

const firebaseConfig = {
    apiKey: "AIzaSyDtX6hC8fxm1MLS3uxvUp6WJVZHzVXG308",
    authDomain: "myfinance-88e2a.firebaseapp.com",
    projectId: "myfinance-88e2a",
    storageBucket: "myfinance-88e2a.appspot.com",
    messagingSenderId: "463500740011",
    appId: "1:463500740011:web:e45c7656f5044cb6ce7eee",
    measurementId: "G-1H0NJWG2MK",
};

if (!firebase.default.apps.length) {
    firebase.default.initializeApp(firebaseConfig);
}

const uploadImagesToFirebase = async (imageUris) => {
    const promises = imageUris.map(async (imageUri, index) => {
        return await uploadImage(imageUri, index);
    });
    return await Promise.all(promises);
};
const uploadImage = async (uri, index) => {
    const response = await fetch(uri);
    const blob = await response.blob();
    const imageName = "images/" + new Date().toString() + index;
    await firebase.default.storage().ref().child(imageName).put(blob);
    return await firebase.default
        .storage()
        .ref()
        .child(imageName)
        .getDownloadURL();
};

export { uploadImagesToFirebase };
```

Slika 15. Učitavanje slika na Firebase

5. MOBILNA APLIKACIJA ZA PRAĆENJE OSOBNIH FINANCIJA

Aplikacija je izrađena koristeći moderne tehnologije za izradu kvalitetnih i oku privlačnih mobilnih aplikacija. Aplikacija se sastoji od izbornika s karticama na dnu ekrana. Prva kartica navodi na glavni ekran aplikacije. Na drugoj kartici nalazi se popis transakcija s mogućnošću filtriranja po datumu i kategorijama. Na trećoj kartici nalazi se linijski grafikon na kojemu je vidljiv trend stanja na računima. Na četvrtoj kartici nalazi se izbornik koji omogućava postavljanje valute, učitavanje i ispis podataka u Excel datoteku, brisanje svih podataka, pregled računa, kategorija i dodatnih polja.

Prvim pokretanjem aplikacije, pojavljuju se ekrani prikazani na slici 16. Prikazani ekrani služe za upoznavanje korisnika s osnovnim funkcionalnostima aplikacije.



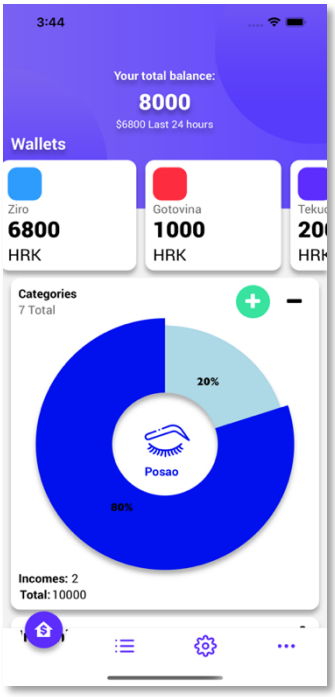
Slika 16. Upute korisnicima

5.1. Registracija i prijava korisnika

Nakon ekrana za upoznavanje korisnika s aplikacijom pojavljuje se ekran dobrodošlice prikazan na slici 17. Na slici u nastavku nalaze se i ekrani za prijavu i registraciju korisnika. Sa ekrana dobrodošlice korisnik može odabrati način rada bez mreže, prijavu ili registraciju u sustav.

Slika 17. Ekrani za prijavu i registraciju korisnika

Nakon prijave korisnika u sustav, aplikacija navodi na glavni ekran aplikacije prikazan na slici 18.

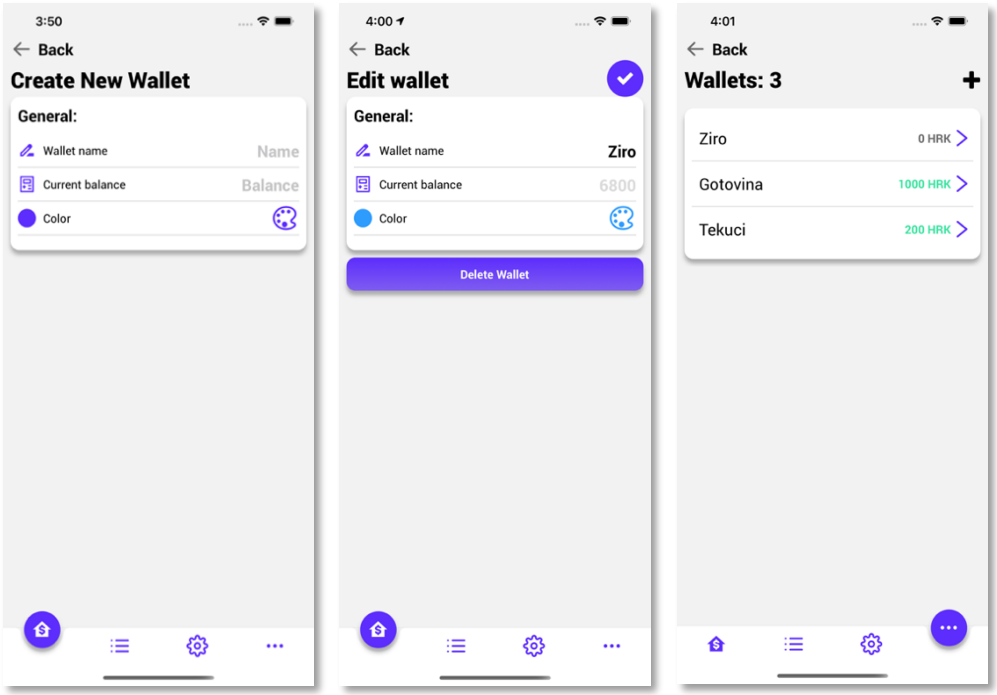


Slika 18. Glavni ekran aplikacije

Na glavnom ekranu aplikacije navedeni su sumarni podaci troškova. Na vrhu ekrana vidljivo je trenutno stanje svih računa, kao i promet u zadnja 24 sata. Tu se nalazi i lista računa sa stanjem na svakom računu. Lista omogućuje dodavanje i uređivanje računa. Na glavnom ekranu nalazi se i tortni grafikon koji prikazuje postotak vrijednosti transakcija po kategorijama. S glavnog ekrana aplikacije moguće je stvoriti nadolazeće transakcije.

5.2. Dodavanje, uređivanje i brisanje računa

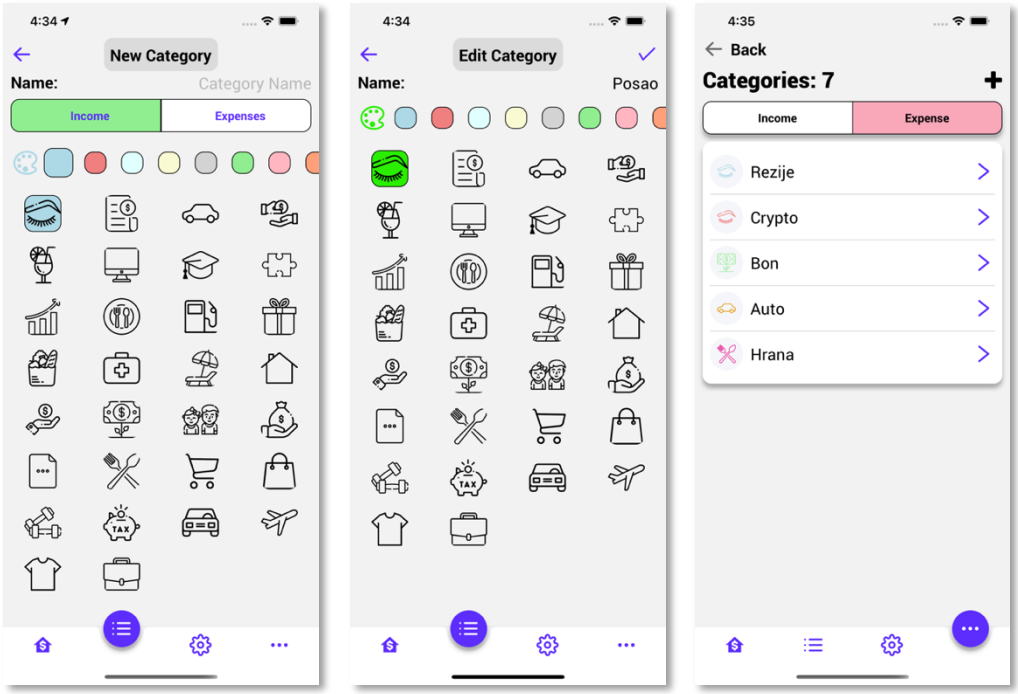
Za stvaranje računa potrebno je unijeti naziv, boju i početno stanje računa. Račun se može obrisati sa ekrana za uređivanje računa i ekrana za prikaz svih računa. Brisanjem računa, brišu se sve transakcije povezanih s računom. Aplikacije dozvoljava stvaranje neograničenog broja računa. Transakcijom tipa „transfer“ prebacuju se sredstva s jednog računa na drugi. Ekrani za dodavanje, uređivanje i brisanje računa prikazani su na slici 19.



Slika 19. Ekrani za dodavanje, uređivanje i pregled računa

5.3. Dodavanje, uređivanje i brisanje kategorija

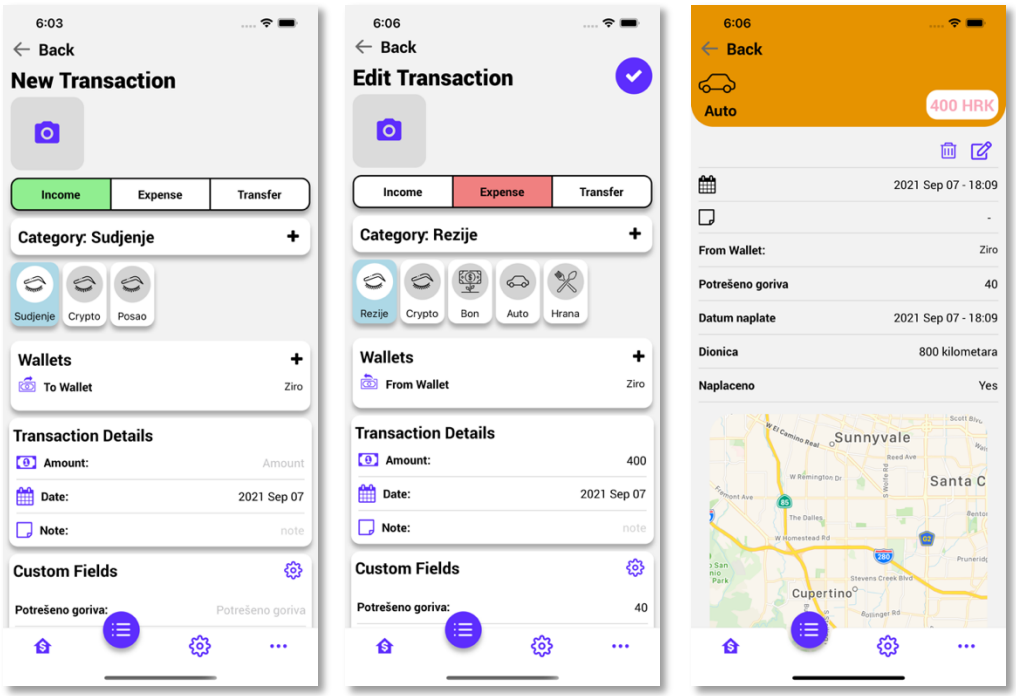
Prije kreiranja transakcija potrebno je kreirati kategorije. Za kreiranje kategorije potrebno je unijeti naziv, odrediti tip, boju i ikonu. Tip kategorije može biti prihod, rashod ili oboje. Brisanjem kategorije brišu se sve transakcije povezane s navedenom kategorijom. Na slici 20. prikazani su ekrani za unos, izmjenu i pregled svih kategorija.



Slika 20. Ekrani za unos, izmjenu i prikaz kategorija

5.4. Dodavanje, uređivanje i brisanje transakcija

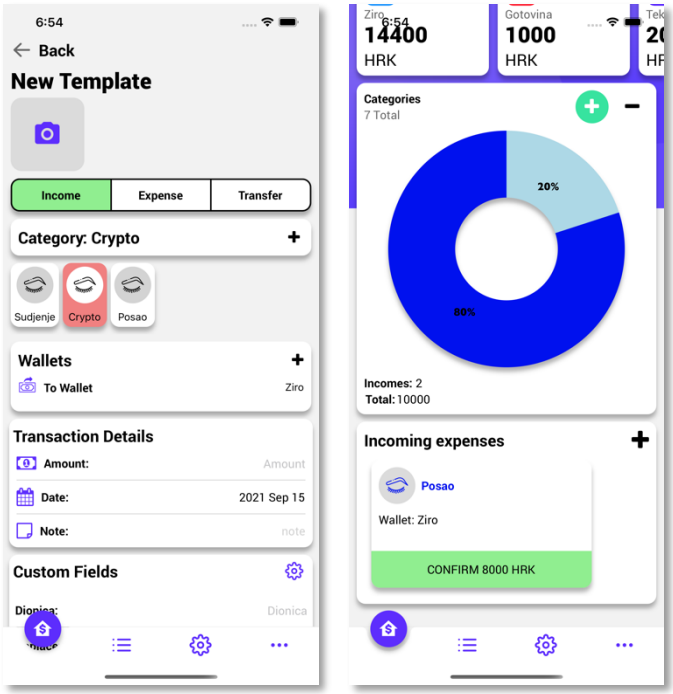
Nakon unosa kategorije, moguć je unos transakcija. Za unos transakcije potrebno je kreirati barem jedan račun i kategoriju. Osim unaprijed definiranih polja transakcijama je moguće definirati dodatna polja prema osobnim potrebama korisnika. Za unos transakcije potrebno je odabrati kategoriju, tip i iznos transakcije. Tip transakcije može biti prihod, rashod ili transfer. Transfer služi za prebacivanje sredstva s jednog računa na drugi. Prilikom unosa transakcije tipa transfer potrebno je definirati iznos, datum i dva računa, račun s kojeg se skidaju sredstva i račun na koji se sredstva prenose. Na slici 21. prikazani su ekrani za unos, uređivanje i prikaz transakcije.



Slika 21. Ekrani za unos, uređivanje i prikaz transakcije

5.5. Kreiranje predložaka budućih transakcija

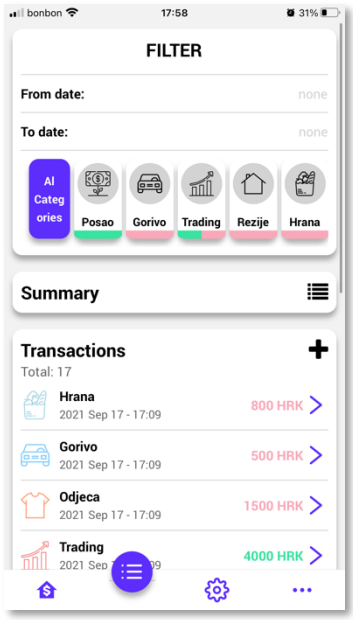
Ključna funkcionalnost aplikacije za koju je potrebno stvoriti posebno korisničko iskustvo je stvaranje novih transakcija. U tu svrhu omogućeno je stvaranje nadolazećih transakcija. Postoje troškovi koji se često ponavljaju i uvijek su jednakog iznosa. Korisniku je omogućeno stvaranje takvih transakcija potvrđivanjem predloška. Na slici 22. s lijeve strane prikazan je ekran za stvaranje nadolazećih transakcija, s desne strane prikazan je glavni ekran aplikacije koji sadrži horizontalnu listu nadolazećih transakcija. Na listi je prikazana jedna nadolazeća transakcija u kategoriji posao u iznosu 8000 kuna.



Slika 22. Ekran za stvaranje i prikaz nadolazećih transakcija

5.6. Filtriranje transakcija

Na drugoj kartici glavnog izbornika nalazi se popis transakcija. Transakcije se filtriraju po datumu i kategorijama. Ekran za prikaz i filtriranje transakcija nalazi se na slici 23.



Slika 23. Ekran za prikaz i filtriranje transakcija

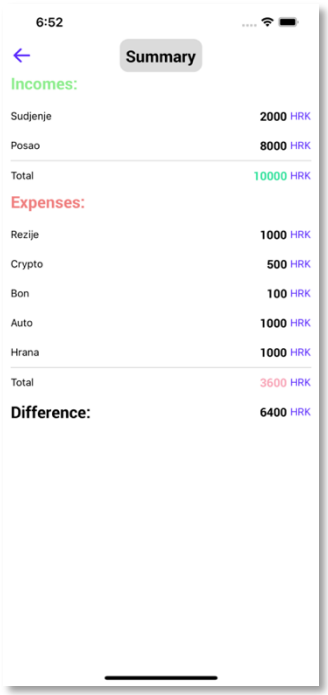
Programski kod za filtriranje transakcija nalazi se na slici 24. Transakcije se filtriraju po datumu, kategoriji i statusu, a zatim su sortirane po datumu.

```
const filteredData = () => {
    let transactions = defaultState.transactions;
    if (fromDate) {
        transactions = transactions.filter(
            (t) => new Date(t.date) >= fromDate
        );
    }
    if (toDate) {
        transactions = transactions.filter(
            (t) => new Date(toDate) >= new Date(t.date)
        );
    }
    transactions = transactions.filter(
        (t) => t.categoryId == category || category == "0"
    );
    transactions = transactions.filter(
        (t) => t.status == constants.processed
    );
    transactions = transactions.sort((a, b) => {
        return new Date(a.date) < new Date(b.date);
    });
    setTransactions(transactions);
};
```

Slika 24. Programski kod za filtriranje transakcija

5.7. Kreiranje obračuna

Na slici 25. prikazan je obračun svih transakcija. Na obračunu se nalazi ukupan promet po kategorijama, zbroj svih prihoda i rashoda te njihova razlika. Generirani obračun može se filtrirati po kategorijama i datumu podešavajući filter na ekranu s popisom transakcija. Na prikazanom obračunu ukupni prihodi iznose 10 000 kuna, ukupni troškovi iznose 3 600 kuna, ukupna razlika između prihoda i rashoda iznosi 6 400 kuna.



Slika 25. Obračun

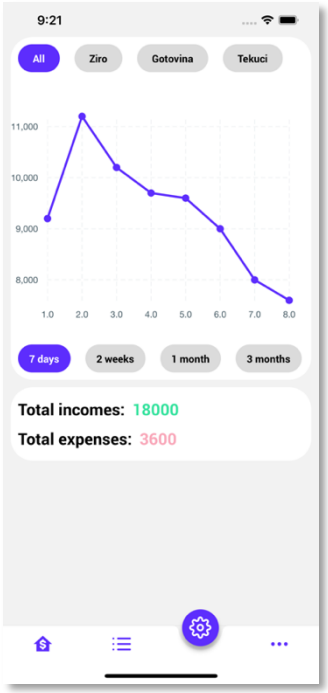
Na slici 26. prikazan je programski kod za pripremu podataka potrebnih za prikaz obračuna. Potrebno je izračunati ukupnu vrijednost transakcija po pojedinoj kategoriji. Za to je potrebno dvije petlje. Vanjska petlja prolazi kroz polje kategorija. Unutarnja petlja prolazi kroz polje transakcija. Za svaku transakciju provjerava se pripada li trenutnoj kategoriji. Ovisno o tipu transakcije ažurira se stanje kategorije. Završetkom unutarnje petlje podaci se spremaju u objekt koji se na kraju vraća iz metode.

```
const SummaryData = (transactions) => {
    const data = {
        incomes: [],
        expenses: [],
        totalExpenses: 0,
        totalIncomes: 0,
        result: 0
    };
    let totalIncomes = 0;
    let totalExpenses = 0;

    defaultState.categories.forEach((category) => {
        let totalIncome = 0;
        let totalExpense = 0;

        transactions.forEach((transaction) => {
            if (transaction.categoryId == category.id) {
                if (transaction.type == constants.income) {
                    totalIncome = +totalIncome + +transaction.amount;
                }
                if (transaction.type == constants.expense) {
                    totalExpense = +totalExpense + +transaction.amount;
                }
            }
        });
        if (totalIncome > 0) {
            data.incomes.push({
                category,
                totalIncome,
            });
        }
        if (totalExpense > 0) {
            data.expenses.push({
                category,
                totalExpense,
            });
        }

        totalIncomes = +totalIncomes + +totalIncome;
        totalExpenses = +totalExpenses + +totalExpense;
    });
    data.totalExpenses = totalExpenses;
    data.totalIncom  let totalIncomes: number
    data.result = +totalIncomes - +totalExpenses;
    return data;
};
```

Slika 26. Programski kod za pripremu podataka obračuna

5.8. Grafički prikaz troškova

Aplikacija omogućava prikaz linijskog grafikona, na linijskom grafikonu prikazuje se kretanje vrijednosti svih računa ili pojedinačnog računa. Grafikon je prikazan na slici 27. Podatke prikazane na grafikonu moguće je filtrirati za protekli tjedan, protekla dva tjedna, zadnjih mjesec dana i zadnja tri mjeseca. Na prikazanom grafikonu vidi se da ukupna vrijednost imovine je narasla preko 11 000 kuna, te je postepeno krenula padati ispod 8 000 kuna. Ispod linijskog grafikona nalaze se ukupni prihodi i rashodi za odabrani period i račun.



Slika 27. Linijski grafikon vrijednosti svih računa

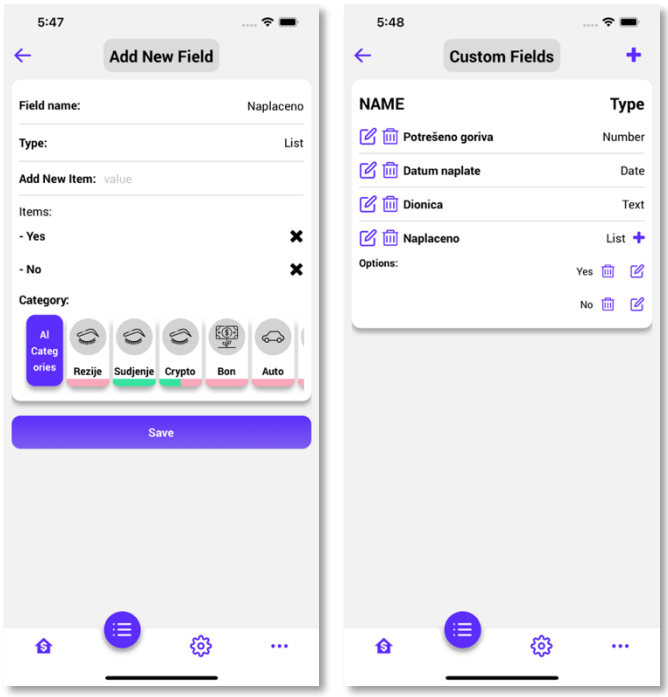
Na slici 28. prikazan je programski kod za pripremu podataka prikazanih na linijskom grafikonu. Potrebno je postaviti početno stanje računa odabranog za prikaz podataka. Transakcije je potrebno filtrirati ovisno o odabranom računu, statusu i datumu. Kada su transakcije za prikaz na grafikonu filtrirane, prolaskom kroz polje transakcija izračunavaju se elementi za prikaz na grafikonu koji se sastoje od vrijednosti stanja na računu.

```
const getChartData = (state, setData, wallet, time) => {
    let initialBalance = 0;
    let selectedWallet = {};
    if (wallet == constants.all) {
        state.wallets.forEach((wallet) => {
            initialBalance = +initialBalance + +wallet.ballance;
        });
    } else {
        let wallets = state.wallets.filter((item) => item.name == wallet);
        selectedWallet = wallets[0];
        initialBalance = selectedWallet.ballance;
    }

    let transactions = [];
    if (wallet == constants.all) {
        transactions = state.transactions;
    } else {
        transactions = state.transactions.filter(
            (a) =>
                a.toAccountId == selectedWallet.id ||
                a.fromAccountId == selectedWallet.id
        );
    }

    transactions = filterTransactionsByDate(transactions, time);
    transactions = transactions.filter((t) => t.status == constants.processed);
    transactions = transactions.sort((a, b) => {
        return new Date(a.date) > new Date(b.date);
    });

    let chartData = [];
    transactions.forEach((transaction, index) => {
        if (wallet == constants.all) {
            if (transaction.type == constants.income) {
                initialBalance = +initialBalance + +transaction.amount;
            }
            if (transaction.type == constants.expense) {
                initialBalance = +initialBalance - +transaction.amount;
            }
        } else {
            if (transaction.toAccountId == selectedWallet.id) {
                initialBalance = +initialBalance + +transaction.amount;
            }
            if (transaction.fromAccountId == selectedWallet.id) {
                initialBalance = +initialBalance - +transaction.amount;
            }
        }
        chartData.push({
            x: index + 1,
            y: initialBalance,
        });
    });
    setData(chartData);
};
```

Slika 28. Programski kod za pripremu podataka prikazanih na linijskom grafikonu

5.9. Kreiranje dodatnih polja

Dodatna polja mogu biti tipa broj, tekst, datum ili lista. Za kreiranje polja tipa lista potrebna su minimalno dva elementa. Polje se može dodijeliti jednoj ili svim kategorijama. Brisanjem dodatnih polja brišu se zapisi u svim transakcijama. Ekran za kreiranje, uređivanje, brisanje i pregled dodatnih polja prikazani su na slici 29.



Slika 29. Ekrani za kreiranje, prikaz i uređivanje dodatnih polja

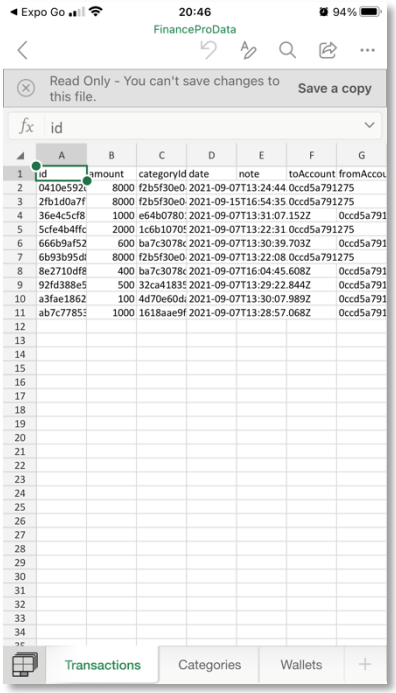
5.10. Čitanje i pisanje podataka u Excel datoteku

Neregistrirani korisnici koji koriste aplikaciju bez korištenja mreže, prilikom zamjene mobilnog uređaja imaju potrebu za prebacivanjem podataka. Kako bi se omogućio prijenos podataka s jednog uređaja na drugi, implementirana je mogućnost pisanja i čitanja podataka koristeći Excel datoteku. Na slici 30. prikazan je programski kod za ispis podataka u Excel datoteku. Za rad s Excel datotekom korištena je „xlsx" biblioteka. Za ispis podataka potrebno je kreirati radnu knjigu „workbook". Zatim se kreiraju radni listovi „work sheets", za svaku tablicu kreira se jedan radni list. Radni listovi dodaju se radnoj knjizi koristeći pomoćnu metodu „book_append_sheet". Pomoću metode „write" podaci se zapisuju u radnu knjigu. „WriteAsStringAsync" metoda iz „expo-file-system" biblioteke sprema Excel datoteku na uređaj.

```
import * as FileSystem from "expo-file-system";  8.1K (gzipped: 2.7K)
import * as Sharing from "expo-sharing";
import XLSX from "xlsx";  891.3K (gzipped: 319.6K)
const exportData = async (state) => {
    var wb = XLSX.utils.book_new();

    var ws = XLSX.utils.json_to_sheet(state.transactions);
    var ws2 = XLSX.utils.json_to_sheet(state.categories);
    var ws3 = XLSX.utils.json_to_sheet(state.wallets);
    var ws4 = XLSX.utils.json_to_sheet(state.customFields);
    var ws5 = XLSX.utils.json_to_sheet(state.customFieldsListValues);
    var ws6 = XLSX.utils.json_to_sheet(state.customFieldsValues);

    XLSX.utils.book_append_sheet(wb, ws, "Transactions");
    XLSX.utils.book_append_sheet(wb, ws2, "Categories");
    XLSX.utils.book_append_sheet(wb, ws3, "Wallets");
    XLSX.utils.book_append_sheet(wb, ws4, "CustomFields");
    XLSX.utils.book_append_sheet(wb, ws5, "CustomFieldsListValues");
    XLSX.utils.book_append_sheet(wb, ws6, "CustomFieldsValues");

    const wbout = XLSX.write(wb, {
        type: "base64",
        bookType: "xlsx",
    });

    const uri = FileSystem.cacheDirectory + "FinanceProData.xlsx";

    await FileSystem.writeAsStringAsync(uri, wbout, {
        encoding: FileSystem.EncodingType.Base64,
    });

    await Sharing.shareAsync(uri, {
        mimeType:
            "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet",
        dialogTitle: "Finance pro data",
        UTI: "com.microsoft.excel.xlsx",
    });
};
```

Slika 30. Programski kod za ispis podataka u Excel datoteku

„shareAsync“ metoda iz „expo-sharing“ biblioteke omogućava otvaranje excel datoteke na mobilnom uređaju. Na slici 31. nalazi se Excel datoteka koja sadrži podatke iz aplikacije.



Slika 31. Excel datoteka s podacima uvezenim iz aplikacije

5.11. Način rada bez mreže

Postoje korisnici koji ne žele plaćati preplate, već su im dovoljne osnovne funkcionalnosti aplikacije. Način rada bez mreže podrazumijeva spremanje podataka korisnika na uređaj. Za rad bez mreže nije potrebna registracija u sustav. U načinu rada bez mreže podaci se spremaju u SQLite bazu podataka. Prilikom pokretanja aplikacije provjerava se korisnik. Ako je registrirani korisnik podaci se dohvaćaju pomoću napravljenog REST API servisa. Ako je ne registrirani korisnik, odnosno način rada bez mreže, podaci se dohvaćaju lokalno sa uređaja. Programski koda za dohvaćanje podataka prilikom pokretanja aplikacije nalazi se na slici u 32.

```
const loadData = async (setState) => {
    await loadStorageData();

    if (defaultState.user == constants.offline) {
        await getDataSQLite();
    } else {
        if (defaultState.token != undefined && defaultState.token != "") {
            const result = await getDataRest();
            saveDataLocalState(result);
        }
    }
    setState({ ...defaultState });
};
```

Slika 32. Programski kod za dohvaćanje podataka

5.12. Brisanje svih podataka

Aplikacija radi s osjetljivim podacima korisnika, zbog toga korisniku je omogućeno brisanje svih podataka iz aplikacije. Na slici 33. prikazan je programski kod za brisanje svih podataka i resetiranje postavki aplikacije. Na početku metode brišu se tablice iz SQLite baze, a zatim kreiraju nove. Ako je registrirani korisnik, brišu se podaci iz MySql baze pozivom REST servisa. Resetiraju se postavke i lokalno stanje aplikacije.

```
const deleteEverything = async () => {
    try {
        dropTablesSQLite();
        createTablesSQLite();

        if (defaultState.user != constants.offline) {
            await deleteUserDataRest();
        }

        await storeData(KEYS.currency, JSON.stringify(currencyData[56]));
        await storeData(KEYS.token, "");
        await storeData(KEYS.user, constants.offline);
        await storeData(KEYS.onBoarding, false);

        resetState();
        defaultState.onBoard = false;
        setState({ ...defaultState });
    } catch (error) {
        showError(error);
    }
};
```

Slika 33. Programski kod za brisanje podataka

6. ZAKLJUČAK

U ovom diplomskom radu izrađena je mobilna aplikacije za vođenje osobnih financija. Kako bi se zadovoljila većina korisnika izrađena je aplikacija za više platformi, iOS i android. Iako je nemoguće udovoljiti željama i potrebama svih korisnika, u izradi ove aplikacije, nastojalo se napraviti generalno rješenje kojim bi se korisnicima omogućilo modificiranje aplikacije, svaki korisnik može definirati dodatna polja koja će unositi u aplikaciju. Navedena funkcionalnost se pokazala kao odlično rješenje za povećanje korisnosti aplikacije i stvaranje komparativne prednosti u odnosu na konkurentska rješenja.

U radu su opisane vrste mobilnih aplikacija, predstavljene su sve tehnologije korištene za izradu kompletnog sustava te je dan primjer njihovog korištenja.

Aplikacija je razvijena koristeći klijent-poslužitelj arhitekturu. Klijentska strana je razvijena koristeći React Native okvir, dok poslužiteljska strana je razvijena Node.js i Express.js bibliotekom. Programski jezici korišteni za razvoj su JavaScript, Sql, HTML i CSS.

Iako proces razvoja aplikacije nikada ne prestaje, može se reći da aplikacija sadrži dovoljno mogućnosti i omogućuje korisnicima detaljan uvid u potrošnju. Daljnjim razvojem aplikacije i implementacijom dodatnih funkcionalnosti kao što su grupno bilježenje troškova, dopisivanje, web verzija i filtriranje dodatnih polja, smatram da bi aplikaciju dovelo na razinu najpopularnijih aplikacija na tržištu.

Trenutno aplikacija ne omogućuje pretraživanje transakcija prema dodatnim poljima korisnika. Aplikacija omogućava rad registriranim i neregistriranim korisnicima. Ako je korisnik registriran omogućen mu je rad samo preko mreže i obrnuto. Trenutni nedostatak aplikacije je što ne omogućava unos podataka registriranom korisniku u slučaju kada nema mreže. Navedeni nedostatak će biti ispravljen u sljedećim verzijama aplikacije.

Ako postoji potreba za izradom moderne mobilne aplikacije temeljene na podacima, korištenje React Native-a na klijentskoj strani u kombinaciji sa Node.js-om na poslužiteljskoj strani daje odlično rješenje. Tim više što React Native privlači veliku pozornost, pa postoje projekti izrađeni oko React Native-a kao što je Expo koji dodatno olakšava razvoj mobilnih aplikacija koje djeluju kao izvorne, a moguće ih je pokrenuti na više platformi kao što su iOS i android.

LITERATURA

- Barbić, D. (nema datuma) ‘Planiranje osobnih financija’, pp. 148–197. dostupno na: https://www.ss-adamic.com/wp-content/uploads/posjet_HNB/IV-Planiranje-osobnih-financija.pdf., [10.08.2021.]
- Brown, E. (2016) *Web development with Node & Express*, *Journal of Chemical Information and Modeling*. dostupno na: https://www.vanmeegern.de/fileadmin/user_upload/PDF/Web_Development_with_Node_Express.pdf., [10.08.2021.]
- Budgetbakers, (2010), dostupno na: https://budgetbakers.com/ [10.08.2021.]
- Crockford, D. (2008) *JavaScript: The Good Parts by Douglas Crockford*. dostupno na: https://andersonguelphjs.github.io/OReilly_JavaScript_The_Good_Parts_May_2008.pdf. [18.08.2021.]
- Eisenman, B. (2016) *Learning React Native: Building Native Mobile Apps with JavaScript*, *Https://Github.Com/React-Native-Community/React-Native-Maps*. dostupno na: https://pepa.holla.cz/wp-content/uploads/2016/12/Learning-React-Native.pdf., [18.08.2021.]
- eTraverse, (nema datuma) dostupno na: https://etraverse.com/article/native-vs-hybrid-vs-cross-platform-what-to-choose-and-when/ [13.08.2021.]
- Firebase, (nema datuma) dostupno na: https://firebase.google.com/docs/storage, [10.08.2021.]
- Maiti i Bidinger (1981) *Reac and React Native*, *Journal of Chemical Information and Modeling*. dostupno na: https://www.rededocancerdelp.com.br/images/transparencia/5c363a069ba2b_reactandreactnative.pdf., [7.09.2021.]
- Manual, M. R. (2013) ‘MySQL 5 . 0 Reference Manual’, *MySQL 5.0 Reference Manual*, 1, p. 1692. dostupno na: dev.mysql.com., [10.08.2021.]
- Luetić M, (2021) dostupno na :https://decode.agency/native-vs-cross-platform-mobile-apps/, [17.08.2021.]
- Saccomani P, (nema datuma) dostupno na: https://www.mobiloud.com/blog/native-web-or-hybrid-apps , [10.08.2021.]

- Stackoverflow, (nema datuma) dostupno na: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language [10.08.2021.]
- Spendee, (2018) dostupno na: https://www.spendee.com/ [10.08.2021.]
- Stančer, D. (2015) *Osnove JavaScripta*. dostupno na: https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/c501_polaznik.pdf. [15.08.2021.]
- Toshl. (2012) dostupno na: https://toshl.com/ , [10.08.2021.]

POPIS SLIKA

Slika 1: Prikaz ekrana za unos transakcija iz aplikacije ToshL Finance 4
Slika 2: Prikaz ekrana za unos transakcija iz aplikacije Spendee 5
Slika 3: Prikaz ekrana za unos transakcija iz aplikacije WalletApp 6
Slika 4: Prikaz najkorištenijih tehnologija u 2021. godini 12
Slika 5: Programski kod za izračunavanje stanja novčanika 13
Slika 6: Glavna komponenta aplikacije 14
Slika 7: Programski kod React Native komponente 16
Slika 8: Struktura izrađenog projekta koristeći Expo 17
Slika 9: Programski kod za pokretanje servera koristeći express.js biblioteku 19
Slika 10: POST akcija za unos transakcija 19
Slika 11: Posrednik za autentifikaciju 20
Slika 12: Zahtjev i odgovor u JSON formatu POST akcije za unos transakcije 20
Slika 13: Struktura baze podataka 22
Slika 14: Pomoćna metoda za rad sa SQLite bazom podataka 24
Slika 15: Učitavanje slika na Firebase 25
Slika 16: Upute korisnicima 26
Slika 17: Ekrani za prijavu i registraciju korisnika 27
Slika 18: Glavni ekran aplikacije 28
Slika 19: Ekrani za dodavanje, uređivanje i pregled računa 29
Slika 20: Ekrani za unos, izmjenu i prikaz kategorija 30
Slika 21: Ekrani za unos, uređivanje i prikaz transakcije 31
Slika 22: Ekran za stvaranje i prikaz nadolazećih transakcija 32
Slika 23: Ekran za prikaz i filtriranje transakcija 33
Slika 24: Programski kod za filtriranje transakcija 33
Slika 25: Obračun 34
Slika 26: Programski kod za pripremu podataka obračuna 35
Slika 27: Linijski grafikon vrijednosti svih računa 36
Slika 28: Programski kod za pripremu podataka prikazanih na linijskom grafikonu 37
Slika 29: Ekrani za kreiranje, prikaz i uređivanje dodatnih polja 38
Slika 30: Programski kod za ispis podataka u Excel datoteku 39
Slika 31: Excel datoteka s podacima uvezenim iz aplikacije 40
Slika 32: Programski kod za dohvaćanje podataka 41

Slika 33: Programski kod za brisanje podataka..... 42

POPIS TABLICA

Tablica 1: Usporedba funkcionalnosti aplikacija 7
Tablica 2: Popis svih metoda REST API-a 21