# Uvod

Tema ovog diplomskog rada je izrada i opis sustava FinAssist, korisničkog osobnog financijskog pomoćnika. Ovaj sustav namijenjen je bilježenju i praćenju korisničkih financijskih transakcija, prihoda i troškova. Njegova glavna sastavnica je aplikacija izrađena za operacijski sustav Android, koju korisnik pokreće preko svog pametnog telefona ili tableta. Koristeći se njome korisnik može unijeti i pohraniti podatke o svakoj svojoj financijskoj transakciji, sa svim relevantnim podacima kao što su novčani iznos, način plaćanja, podaci o uplatitelju i primatelju i sl. Aplikacija također omogućava pregled podataka o korisnikovim transakcijama i transakcijskim računima u željenom vremenskom razdoblju, te uz ostale parametre pretrage i filtriranja.

Podaci o transakcijama ne spremaju se lokalno, već u udaljenu bazu podataka. Za to se koriste backend servisi koje pruža razvojna platforma Firebase. Integracija te platforme pruža pristupno sučelje preko kojeg se podaci aplikacije mogu spremati u „oblaku“ (engl. cloud) u formi NoSQL baze podataka, te se njima može upravljati preko CRUD funkcionalnosti (engl. create, read, update, delete - stvaranje, dohvaćanje/čitanje, izmjena, brisanje), koje korisnik poziva preko sučelja implementiranog u klijentskoj aplikaciji. Također, za potrebe zaštite tajnosti i povjerljivosti korisničkih podataka, Firebase omogućava implementaciju odgovarajućeg sučelja za korisničku autentifikaciju.

# Korisnički i zahtjevi sustava

## Funkcionalni zahtjevi

Osnovna funkcionalnost koja se očekuje od FinAssist sustava je pohrana podataka o korisničkim financijskim transakcijama i upravljanje tim podacima. To znači da bi klijentska aplikacija trebala pružati sučelje koje korisniku omogućava unos svih relevantnih podataka o obavljenoj transakciji. Ti podaci obavezno moraju uključivati podatke o stranama u transakciji (uplatitelju i primatelju), vrijeme kada je transakcija zabilježena, te novčani iznos transakcije. Nakon što se transakcija spremi, unutar aplikacije moraju biti implementirane funkcionalnosti izmjene podataka o prijašnjim transakcijama, te njihovo brisanje. Također, kako bi se korisniku olakšao i poboljšao uvid u njegove transakcije, one moraju biti pretražive prema svim svojim podacima, i na temelju korisničke pretrage mora postojati mogućnost filtriranog pregleda transakcija.

Analogno upravljanju podacima o transakcijama, radi bolje podatkovne organizacije nužna je mogućnost izravnog upravljanja transakcijskim računima, tj. stranama u financijskoj transakciji. Korisniku mora biti omogućeno da unosi podatke ne samo o računima koje sam koristi u transakcijama, već i o drugim stranama koje se pojavljuju u njegovim transakcijama. Podaci o transakcijskim računima čiji unos i pohrana moraju biti podržani su naziv i trenutna bilanca (financijsko stanje računa). Kao i kod transakcija, funkcionalnosti pregleda, pretrage i filtriranja, te naknadne izmjene i brisanja podataka o računima moraju biti podržane.

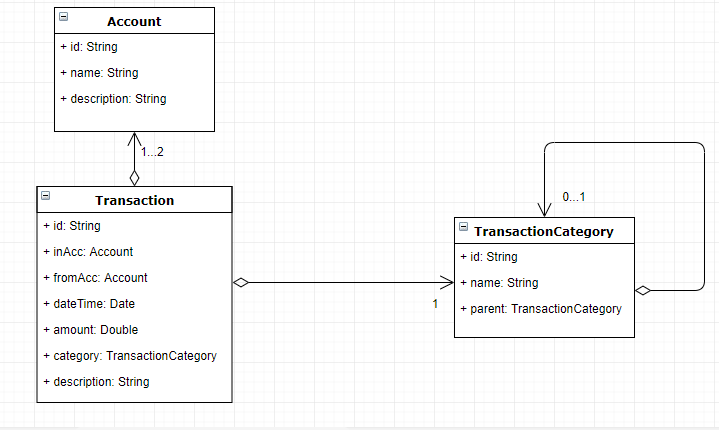
Konačno, radi očuvanja sigurnosti i tajnosti korisničkih podataka mora biti implementiran neki vid autorizacije i autentifikacije, čime će svaki korisnik aplikacije imati pristup i biti ograničen na rad sa samo onim podacima koje je sam unio korištenjem aplikacije.

## Nefunkcionalni zahtjevi

Kako bi aplikacija bila prihvatljiva što većem broju korisnika, poželjno je da korisničko sučelje bude jasno i intuitivno, uz adekvatan vizualni dizajn i bez suvišnih informacija koje bi opterećivale korisnika, te da se uz minimalni broj koraka mogu obaviti sve željene akcije. Budući da se radi o obrascima uporabe koji su iz korisničke perspektive kratki i jednostavni, poželjno je i da njihovo izvršavanje unutar aplikacije bude brzo, što je moguće postići kvalitetnom implementacijom uz minimizaciju dohvaćanja podataka iz memorije ili baze podataka. Druge moguće smjernice pri razvoju orijentiranom k povećanju broja korisnika su smanjenje potrebne količine radne memorije (RAM), te izbjegavanje nepotrebnih funkcionalnosti iz kasnijih verzija sustava Android, čime bi se omogućilo korištenje aplikacije i na tehnički slabijim Android uređajima.

Spremanje podataka na udaljenu bazu podataka umjesto na sam uređaj još je jedna poželjna značajka, budući da na taj način ne samo da se izbjegava bespotrebno trošenje memorijskog prostora na mobilnom uređaju, već i korisniku pruža sigurnost od gubitka podataka uslijed kvara ili gubitka samog uređaja. To, međutim, zahtjeva internetsku vezu pri upravljanju podacima, kao i odgovarajući backend poslužiteljski sustav za upravljanje zahtjevima klijentske aplikacije, tako da je u tom slučaju vrlo važno optimizirati, odnosno minimizirati količine podataka koje aplikacija izmjenjuje s udaljenim poslužiteljem, kako do ne bi došlo do nepotrebnog trošenja korisnikovog ograničenog podatkovnog prometa.

# Model

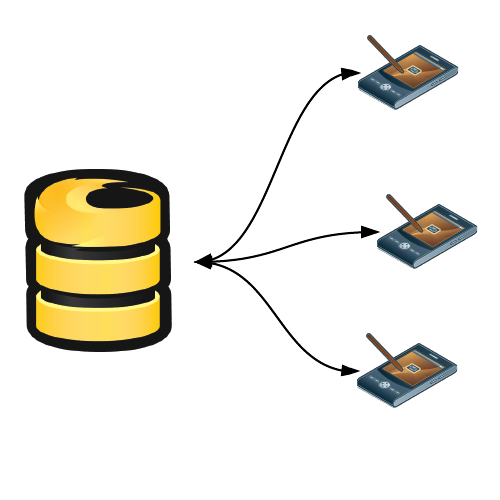


Dva osnovna razreda u modelu koji aplikacija koristi su razredi Account i Transaction. Razred Account mogu se opisuje račune koje korisnik može upotrijebiti za svoje financijske transakcije, kao što su tekući ili žiro račun, ili drugi načini plaćanja kao što je gotovinsko plaćanje. Također, korišten je za opis drugih strana u transakciji, od kojih korisnik transakcijom može primiti novac, ili ga njima uplatiti. Ovaj razred se sastoji od naziva računa i opcionalnog opisa.

Razred Transaction bilježi podatke o provedenim financijskim transakcijama. Svaka transakcija sadrži podatke o korisnikovim računima koji su uključeni u transakciju, ovisno o tipu transakcije, te o novčanom iznosu transakcije. Dodatni podaci su tip transakcije (uplata, isplata, ili transfer – ovdje se misli na prijenos s jednog od korisnikovih računa na drugi) vremenska oznaka, koja se automatski zapisuje uz svaku transakciju, te informacije o transakciji; opcionalan opis, te kategorija. Transakcijska kategorija, opisana razredom TransactionCategory podržava strukturu stabla, što znači da svaka kategorija, osim naziva, može imati svog „roditelja“, tj. nad-kategoriju (npr. nad-kategorija „plaća“ može imati pod-kategorije „redovna plaća“ i „bonus“).

# Arhitektura

## Općenito



(TODO)

## Firebase

Za razvoj poslužiteljske strane FinAssist aplikacije korištena je platforma Firebase, koja omogućuje razvojnim programerima velik broj usluga korisnih za implementaciju i korištenje u mobilnim i web aplikacijama. Firebase je 2011. započeo kao Envolve, projekt koji je pružao API za integraciju *chat* funkcionalnosti u web stranice, no nakon što su primijetili da se njihova usluga koristi za sinkronizaciju drugih aplikacijskih podataka, njegovi osnivači su ga odlučili proširiti i prenamijeniti, što je rezultiralo eventualnim preuzimanjem od strane Googlea 2014. i transformacijom u unificiranu platformu za razvoj mobilnih aplikacija.

Firebase usluge koje aplikacija koristi su Authentication, koja pruža mogućnost autentifikacije korisnika i sustav za upravljanje postojećim korisnicima, i Realtime Database, koja omogućava spremanje podataka u NoSQL bazu i sinkronizaciju svih klijenata s tim podacima u stvarnom vremenu. Neke od drugih korisnih Firebase usluga su Cloud Messaging, multiplatformska usluga za razmjenu poruka i obavijesti, Storage za spremanje datoteka, te Analytics i Crashlytics za praćenje aktivnosti i korištenja aplikacija, kao i generiranje detaljnih izvješća o greškama u radu aplikacija.

## Firebase Authentication

Firebase Authentication pruža podršku za identifikaciju i autentifikaciju korisnika aplikacije. Poznavanje korisnikovog identiteta aplikaciji omoguće spremanje korisničkih podataka i pružanje istog personaliziranog iskustva na više različitih uređaja. FinAssist aplikacija podržava autentifikaciju preko e-mail adrese i lozinke, a Firebase Authentication inače omogućava i druge načine autentifikacije, npr. preko Facebook ili Google korisničkog računa.

Na početku korištenja aplikacije korisnik mora stvoriti novi korisnički račun unosom željenog e-maila i lozinke. Nakon provjere valjanosti unesenih podataka, šalje se zahtjev za stvaranje računa metodom createUserWithEmailAndPassword koja kao parametre prima te podatke. Zatim se postavlja slušač (eng. listener) koji čeka završetak te radnje i, ovisno o uspješnosti stvaranja korisničkog računa, omogućava korištenju daljnje, autorizirano korištenje aplikacije ili dojavljuje grešku.

mAuth.createUserWithEmailAndPassword(email, password)  
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
            @Override  
            public void onComplete(@NonNull Task<AuthResult> task) {  
                if (task.isSuccessful()) {  
                    // Stvaranje korisničkog računa uspješno  
                } else {  
                    // Stvaranje korisničkog računa neuspješno  
                }  
            }  
        });

Kasnija autentifikacija (sign-in) je vrlo slična, osim što se koristi metoda signInWithEmailAndPassword (s istim ulaznim parametrima, e-mailom i lozinkom).

Dohvaćanje podataka o autentificiranom korisniku je moguće korištenjem razreda FirebaseUser i metode getCurrentUser().

FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();  
if (user != null) {  
    // Korisnik je uspješno autentificiran  
    String email = user.getEmail();  
}

## Firebase Realtime Database

Firebase Realtime Database je NoSQL baza podataka zasnovana na spremanju podataka „u oblaku“ (engl. cloud storage). Podaci u njoj se spremaju u tekstualnom JSON obliku i sinkronizirano osvježavaju na svakom povezanom klijentu u stvarnom vremenu (engl. realtime). Odlike ove Firebase usluge su:

* Ažuriranje u stvarnom vremenu – umjesto tipičnih HTTP zahtjeva i REST API sučelja, primarni način komunikacije klijenata s bazom je sinkronizacija podataka. To znači da pri svakoj promjeni podataka, svaki povezani uređaj gotovo trenutno dobiva obavijest o toj promjeni.
* Izvanmrežni (engl. offline) rad – čak i ako veza s bazom trenutno nije dostupna, aplikacija ostaje funkcionalna, jer se podaci privremeno spremaju u memoriju uređaja. Nakon što se veza ponovno uspostavi, izvršava se sinkronizacija podataka.
* Pristup s klijentskih uređaja – nema potrebe za poslužiteljskom aplikacijom; podacima u bazi pristupa se izravno s mobilnog uređaja ili web preglednika. Sigurnosne postavke i validacija podataka dostupne su kroz Firebase Realtime Database Security Rules, skup sigurnosnih pravila koja se provjeravaju pri svakom čitanju ili unosu podataka.

Budući da se ne radi o relacijskoj bazi, Realtime Database ne organizira podatke prema klasičnoj tabličnoj shemi; koncepti kao što su SQL upiti i strani ključevi nisu izravno podržani. Umjesto toga koristi se dokument format NoSQL baza podataka; podaci se spremaju u tekstualnom obliku u JSON stablo. Primjer takve organizacije podataka u FinAssist aplikaciju, prikazan je na slici.



U ovakvoj strukturi korijen stabla je tzv. „instanca“ baze koju koristi aplikacija, a svaki novi podatak sprema se kao čvor u tom stablu. Baza podataka FinAssist organizirana je tako da se na drugoj razini nalaze kategorije objekata (zapravo razredi), kao što su transakcije i transakcijski računi, na trećoj korisnici aplikacije kojima pripadaju ti objekti, te na četvrtoj sami objekti, dok se na dubljim razinama nalaze pojedine komponente tih složenih objekata, ako postoje. Čvorovi koji predstavljaju korisnike i korisničke objekte predstavljeni su svojim jedinstvenim identifikatorima, koji su im dodijeljeni prije spremanja u bazu; korisničkom identifikatoru dodan je korisnikov e-mail radi bolje preglednosti.

### Spremanje podataka u bazu

Za spremanje novog podatka potrebna je referenca na čvor kojem će kao dijete biti dodan čvor s novim podacima, koju je moguće izgraditi korištenjem metoda getReference() i child(). Zatim se korištenjem push().getKey() dobiva jedinstveni identifikator objekta, koji se dodjeljuje tom objektu. Konačno, metodom setValue() objekt se sprema u bazu. U primjeru je prikazano dodavanje novog transakcijskog računa, no postupak je analogan i za objekte ostalih razreda.

final FirebaseDatabase database = FirebaseDatabase.getInstance();  
DatabaseReference dbAccounts = database.getReference("accounts“ );

String objectId = dbAccounts.child(userId).push().getKey();

account.setId(objectId);

dbAccounts.child(userId).child(objectId).setValue(account);

// Konačna lokacija novog objekta:

// $database/accounts/$userId/$objectId

### Čitanje podataka iz baze

Za razliku od spremanja u bazu, ne postoji metoda analogna setValue() koja bi omogućavala izravno čitanje iz baze. Umjesto toga potrebno je definirati slušač koji će reagirati na promjene podataka u određenom čvoru ili podstablu. Postoji nekoliko vrlo sličnih načina za to, a jedan od njih prikazan je u donjem primjeru. Ovdje se slušač dodaje samo za prvi događaj promjene podataka, a budući da se metoda onDataChange() poziva i kad je slušač prvi put pridružen čvoru, ovakav slučaj je zapravo neizravna implementacija (nepostojeće) metode getValue() za dohvaćanje svih elemenata djece tog čvora.

ValueEventListener accountListener = new ValueEventListener() {  
    @Override  
    public void onDataChange(DataSnapshot dataSnapshot) {  
        for (DataSnapshot data : dataSnapshot.getChildren()) {  
        Account account = data.getValue(Account.class);  
    accountList.add(account);  
    }  
  
    @Override  
    public void onCancelled(DatabaseError databaseError) {  
        // Čitanje nije uspjelo  
    }  
};  
dbAccounts.child(userId).addListenerForSingleValueEvent(accountListener);

Ukoliko je potrebno stalno pratiti promjene u podstablu nekog čvora, umjesto višestrukog dodavanja istog slušača gore korištenom metodom addListenerForSingleValueEvent() moguće je trajno (tj. do ručnog uklanjanja) dodati slušač na čvor metodom addValueEventListener(). Međutim, mana slušača ValueEventListener je to što nakon svake promjene u podstablu čvora vraća svu djecu tog čvora, što može rezultirati slanjem nepotrebno velike količine podataka. Zbog toga je u tu svrhu bolje koristiti ChildEventListener koji može reagirati na dodavanje, promjenu ili brisanje pojedinog djeteta promatranog čvora, što znači da može dohvaćati samo relevantne podatke (tj. samo onaj čvor čiji su se podaci promijenili).

# Klijentska aplikacija – Android

(TODO)