IMPLEMENTACIJA PROGRAMSKOG RJEŠENJA ZA VOĐENJE OSOBNIH FINANCIJA

Fikajz, Borna

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IMPLEMENTACIJA PROGRAMSKOG RJEŠENJA ZA VOĐENJE OSOBNIH FINANCIJA

Borna Fikajz

Zagreb, veljača 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 28. veljače 2023.

Predgovor

Želio bih se ponajprije zahvaliti svojim roditeljima na ljubavi, podršci i strpljenju tijekom svoga studija. Bez njihova zalaganja, ne bih bio u mogućnosti studirati niti raditi u struci koja me zanimala još odmalena.

Zahvalio bih se svojim kolegama i timu u tvrtki Photomath na njihovom razumijevanju za moje studijske obaveze te na njihovim brojnim savjetima koji su me vodili tijekom izrade ovog rada.

Naposljetku, želio bih se zahvaliti svome mentoru Danielu Beleu na njegovu mukotrpnom radu s nama studentima na Algebri. Njegov je pristup nastavi uvijek i mene i moje kolege poticao da dublje „kopamo“ za znanjem i tražimo više. Svojim nas je primjerom potaknuo da se usudimo posegnuti za svojim snovima.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Borna Fikajz, JMBAG 0321011828, OIB 14039889948 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 11.02.2022. godine, ima odobrenu izradu završnog rada s temom: **Implementacija programskog rješenja za vođenje osobnih financija**

i sažetkom rada: Student će svojim radom prikazati napredno korištenje inovativnih tehnologija u kontekstu razvoja sustava za upravljanje osobnim financijama. Rezultat ovog završnog rada je sustav koji omogućuje interaktvino i napredno koristenje tehnologije u svrhu upravljanja novcem.

Mentor je: Daniel Bele.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

Tema ovog rada je predstavljanje problema vođenja osobnih financija u 21. stoljeću i postavljanje jednoga potencijalnog programskog rješenja. Rezultat je rada mobilna aplikacija za operacijski sustav android koja kroz moderno i intuitivno sučelje omogućava korisnicima pregled i upravljanje svojim osobnim budžetom. Korisnicima se nude mogućnosti praćenja stanja računa, budžetiranja, praćenja stanja tržišta dionica i kriptovaluta te nekolicina manjih mogućnosti. Ovim rješenjem nastoji se umanjiti broj mladih koji imaju poteškoće s vođenjem osobnih financija i osvijestiti što veći broj ljudi o važnosti financijske pismenosti. Za izradu sustava korištena je PostgreSQL baza podataka, Ktor programski okvir za izradu web-servisa te Jetpack Compose programski okvir za izradu android aplikacija.

Ključne riječi: financijska pismenost, budžet, android, aplikacije, PostgreSQL, Ktor, Jetpack Compose

Abstract

The main topic of this paper is presenting the problem of managing personal finances in the 21st century. The result of this work is a mobile application for the Android OS which enables users to review and manage their personal budget by using a modern and intuitive user interface. The users may check the status of their bank account, create budgets, track the state of the stock and cryptocurrency markets along with more smaller features. The goal of this solution is to decrease the number of young adults facing problems with personal finances and spread awareness about the importance of financial literacy. The project was implemented using PostgreSQL as the database, the Ktor framework for creating web services and the Jetpack Compose framework was used for creating the mobile app.

Keywords: financial literacy, budget, Android, applications, PostgreSQL, Ktor, Jetpack Compose

Sadržaj

1. Uvod

Već od doba prvih ljudskih zajednica u paleolitiku naši su preci morali savladati vještinu raspolaganja resursima u uvjetima gdje je najmanja pogreška mogla značiti razliku između života i smrti. Čovječanstvo živi u realnosti ograničenih resursa, gdje je sposobnost razmatranja vrijednosti resursa i pomno planiranje njihova korištenja ključno za osiguranje budućnosti. Možda se više ne bavimo trampom, dukate i sol zamijenile su kreditne i debitne kartice, a svoj kapital više ne skrivamo u madrace, već ga ulažemo u nekolicine financijskih konstrukcija, ali ta osnovna vještina baratanja vlastitim imetkom ostala je ista ili čak postala znatno kompleksnija.

Financijska pismenost jedna je od vještina koja se, barem kod nas u Hrvatskoj, slabo razvija kod djece, a u školama nemamo formalne predmete kojima ih podučavamo o značenju i vrijednosti novca. Koliko god tu vještinu zanemarivali, vrlo brzo u svačijem životu dođe vrijeme kada ona postaje jedan od najosnovnijih alata koje moramo primjenjivati u svakodnevici.

Mnogi studenti prilikom odlaska na studij prvi put preuzimaju vlastite financije u svoje ruke. Naučiti raspolagati svojim novcem bilo bi znatno lakše koristeći digitalne alate na našim, sada već sveprisutnim, pametnim uređajima.

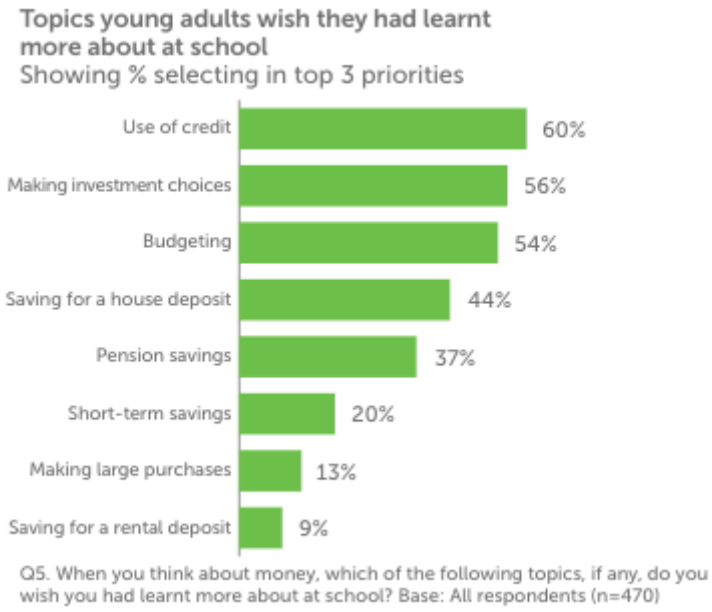
Cilj je ovoga rada upravo predstaviti problem vođenja osobnih financija kod mladih osoba, proučiti trenutno dostupna digitalna rješenja i predstaviti jedno osobno, potencijalno rješenje. U prvome poglavlju proučava se statistika financijske pismenosti među mladima i uspoređuju dva najpopularnija digitalna alata za budžetiranje na tržištu. U drugom poglavlju predstavit će se arhitektura inovativnoga programskog rješenja za budžetiranje i istražiti neke od najboljih praksi za izgradnju web-servisa i mobilnih aplikacija. Nakon toga, proći će se detalji same implementacije projekta i vidjeti neki od izazova koji su se pojavili tijekom izrade. Također, predstavit će se sigurnosni principi kojima nastojimo zaštititi podatke korisnika sustava. Naposljetku, istražit će se važnost i dobri principi prikupljanja analitike korištenja mobilnih aplikacija te potencijalne daljnje nadogradnje za predstavljeno rješenje.

2. Problematika vođenja osobnih financija i prijedlog rješenja

U ovom će se poglavlju detaljno predstaviti problem vođenja osobnih financija, nabrojat će se i usporediti neka od popularnijih, javno dostupnih rješenja i, na kraju, proučiti vlastito inovativno rješenje.

2.1. Obrazloženje problema

Kao što je ranije navedeno, globalno financijsko uređenje nije nikada bilo kompleksnije. U modernom životu baratamo gotovinom, karticama, nekretninama, zlatom, dionicama, kriptovalutama i raznim drugim oblicima kapitala. Znati vješto baratati svojim financijama jedna je od važnijih životnih vještina. Nažalost, kako je prethodno spomenuto, mlade generacije ne podučavamo ovoj vještini niti približno onoliko koliko bismo trebali. Većina mladih susreće se sa svijetom financija tek od trenutka osamostaljenja, kada ova vještina postaje neophodna.
Agencija BritainThinks u suradnji s britanskom službom za financijsko savjetovanje 2017. godine provela je istraživanje o financijskoj pismenosti mladih srednjoškolske i studentske dobi [1]. Anketiranjem je utvrđeno da 61% mladih smatra da bi njihov život bio bolji da posjeduju bolju vještinu raspolaganja novcem. Samo 12% ispitanika smatralo je da su kroz školovanje stekli zadovoljavajuću razinu financijske pismenosti.



Slika 2.1 Prioritetne teme financijske pismenosti mladih u UK

Izvor: Money and pensions service UK [1]

54% ispitanika odgovorilo je da su željeli naučiti više o budžetiranju u školi, a 56% željelo je naučiti više o odabiru ulaganja. Kao što je vidljivo, mladi su svjesni iznimne važnosti dobroga gospodarenja novcem te su svjesni propusta u svojem školovanju po pitanju ove teme.

U svome istraživanju, Gaurav Sinha, Kevin Tan i Min Zhan, sa sveučilišta u Illinoisu, ispitali su 3050 mladih u dobi od 18 do 24 godine o njihovim socio-ekonomskim prilikama i razini financijske pismenosti. Rezultati njihova anketiranja pokazuju da se 22% ispitanika smatra financijski stabilnim, 36% ispitanika smatra se financijski rizičnom skupinom, 32% ispitanika ulazi u grupu financijski nesigurnih, dok 10% ispitanika ulazi u grupu financijski natprosječnih. Kod grupe financijski stabilnih osoba pokazao se trend viših razina financijske pismenosti. Zanimljiv je podatak da su grupe, koje su pokazale najniži stupanj financijske pismenosti, upravo najimućniji i najsiromašniji ispitanici. Dakle, čak i financijski natprosječno stojeći mladi nisu upoznati s načinima pravilnog raspolaganja vlastitim novcem[2].

Vidljivo je da znanstvena istraživanja potvrđuju činjenicu da velik broj mladih ljudi ima poteškoća s gospodarenjem svojim kapitalom. Već spomenuta britanska Agencija za financijsko savjetovanje upravo kao jedan od svojih osnovnih financijskih savjeta predlaže korištenje digitalnih rješenja za pomoć pri financijskom planiranju. Kroz sljedeće poglavlje ukratko će se predstaviti neka od popularnih, trenutno dostupnih rješenja za vođenje osobnih financija.

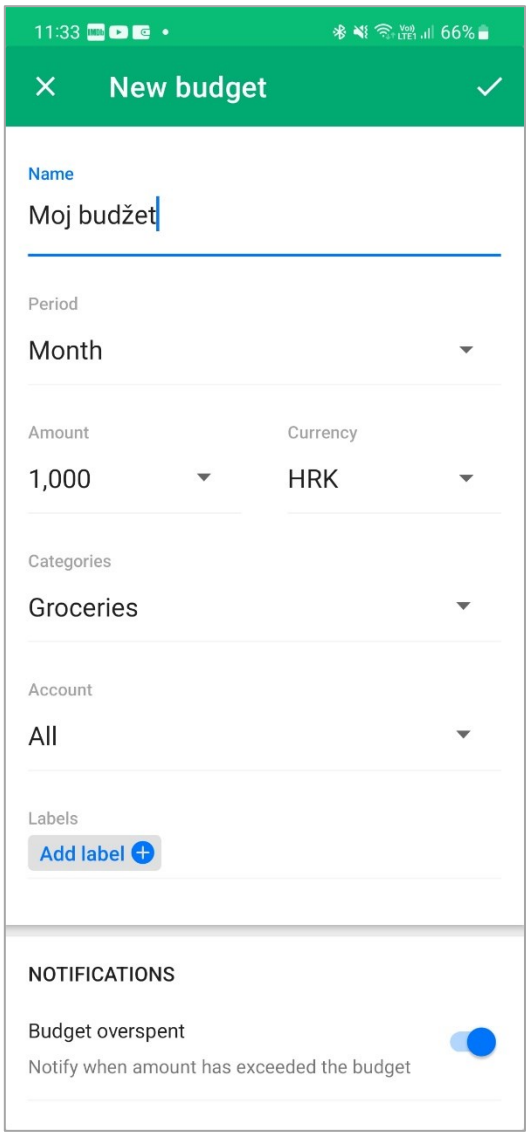
2.2. Pregled postojećih rješenja

Tijekom prošlog desetljeća pametni su telefoni uistinu zavladali svijetom, a potrošačima su omogućili korištenje raznovrsnih digitalnih alat, praktički iz vlastita džepa. Aplikacije za mobilne telefone olakšale su živote raznim kategorijama ljudi svake životne dobi. Od medicinskih aplikacija za dijabetičare, fitness aplikacija za sportaše pa sve do navigacijskih aplikacija za ljubitelje prirode, naizgled mogućnosti digitalizacije alata nemaju kraja. Digitalizacija nije izbjegla ni sektor osobnih financija stoga na tržištu postoji izvjestan broj rješenja za pomoć pri vođenju osobnog budžeta. U ovom će se poglavlju dati kratak pregled nekih poznatijih rješenja i usporedit će se njihove mogućnosti.

2.2.1. Wallet

Wallet je mobilna aplikacija za vođenje osobnih financija koju razvija češka tvrtka BudgetBakers, stacionirana u Pragu. BudgetBakers je češki *fintech startup*, osnovan 2010. godine, koji u svom portfoliju nudi još jedan financijski alat namijenjen za vođenje budžeta na projektima. Wallet je lansiran 2011. godine na platformama android i iOS, a kasnije je stvorena i web-aplikacija. Wallet je fokusiran na dva primarna zadatka: praćenje stanja računa i gotovine te definiranje budžeta. Korisnicima omogućava ručno unošenje stanja gotovine koju trenutno posjeduju, a nudi i opciju povezivanja bankovnog računa. Opcije budžetiranja podijeljene su na dvije kategorije: budžete i ciljeve. Budžeti su osmišljeni kao tradicionalni sustav raspodjele novca za vremensko razdoblje. Prilikom stvaranja budžeta korisnik može definirati vremensko razdoblje za taj budžet, dodijeliti ga u jednu od predodređenih kategorija, namjestiti iznos budžeta, vezati ga uz jedan od spojenih računa i postaviti alarme za prekoračenje. Definiranje ciljeve predstavlja koncept praćenja štednje. Prilikom kreiranja cilja korisnik može definirati kategoriju cilja, dati mu opis, postaviti iznos koji planira uštedjeti te postaviti vremenski rok do kada planira uštedjeti navedenu

svotu. Ciljevi se ručno podešavaju pa korisnik mora unijeti iznos novca koji je odvojio za svoj cilj. Važno je napomenuti da napredne opcije, poput pregleda grafova potrošnje, izrade izvještaja i povezivanje bankovnih računa, nisu dostupne u besplatnoj inačici. Za pristup ovim mogućnostima potrebno je uzeti premium pretplatu. Moguće je uzeti mjesečnu pretplatu za 3,48$, godišnju za 41,76$ i doživotni plan za 83,52$.

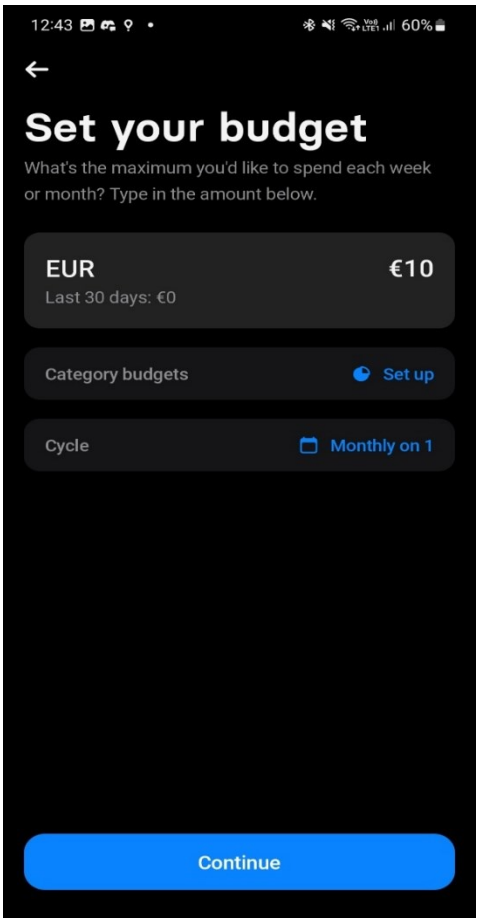


Slika 2.2 Mogućnosti izrade budžeta u aplikaciji Wallet

2.2.2. Revolut

Revolut je aplikacija za mobilno bankarstvo koju razvija istoimena *fintech* tvrtka stacionirana u Londonu. Revolut je osnovan 2015. godine kao odgovor na frustracije svojih osnivača na spor i problematičan proces digitalizacije usluga kod tradicionalnih banaka.

Revolut je puno više od aplikacije za budžetiranje pa je i kao tvrtka orijentiran prema stjecanju bankarskih licencija i direktnom konkuriranju postojećim bankama. Revolut svojim korisnicima nudi: stvaranje digitalnih debitnih kartica, izvršavanje transakcija prema drugim računima, kupovinu kriptovaluta i dionica, kupovinu paketa putnog osiguranja, izradu budžeta povezanih s digitalnim računima, otvaranje štednih računa i još mnoštvo manjih mogućnosti. Mogućnosti vođenja potrošnje zasnovane su na detaljnom pregledu potrošnje za određeno razdoblje, pregledu potrošnje po kategorijama i zakazivanju budućih plaćanja. Kreiranje budžeta omogućava korisniku unošenje iznosa budžeta, njegovo postavljanje u određenu kategoriju te određivanje vremenskog ciklusa za koji budžet vrijedi. Jedina ograničenja kod Revolutovog budžetiranja su nemogućnost praćenja gotovine kao odvojenog izvora (gotovina postoji samo kao potkategorija budžeta) i činjenica da sve ove napredne opcije budžetiranja djeluju jedino ako korisnik koristi Revolut kao primarnu banku. Transakcije nepovezane s Revolutom nije moguće pratiti pa je korisnik primoran svoje bankarske potrebe tražiti od Revoluta. Revolut je u osnovnoj inačici besplatan, a mogućnosti budžetiranja su dostupne i s osnovnim paketom.



Slika 2.3 Mogućnosti izrade budžeta u Revolut aplikaciji

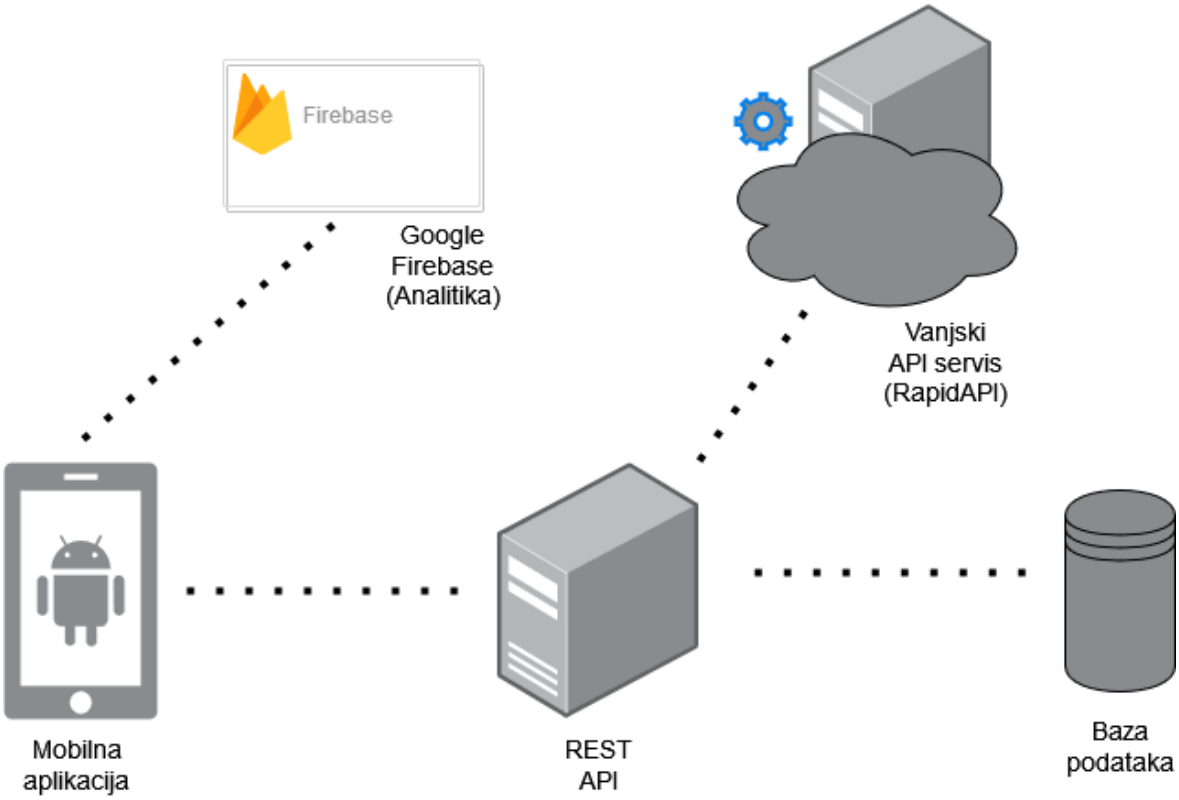
Njegovi plaćeni paketi vezani su uz povećanje limita kod transfera, veći broj digitalnih računa, izradu ekskluzivnih debitnih kartica. Paket Plus korisnicima za 3 € mjesečno nudi zaštitu pri online kupovini, povrat novca za ulaznice i postavljanje računa za maloljetnike. Premium paket za 7 € mjesečno dodaje i opcije neograničene konverzije valuta i više limite za beznaknadno podizanje novca na bankomatima. Najskuplji paket Metal, za 13 € mjesečno, nudi najviše limite, ekskluzivne debitne kartice, uključeno putno osiguranje i veći broj manjih pogodnosti u inozemstvu.

2.3. Prijedlog optimalnog rješenja

Prijedlog za rješenje ovog problema je mobilna aplikacija koja omogućava korisnicima da na jednome mjestu imaju unificirani pregled svih svojih financija. Ona bi omogućavala praćenje i gotovine i transakcija s bankovnih računa. Idealno, sustav aplikacije ne bi imao uvid u korisnikove financijske podatke, već bi ih samo prosljeđivao od informatičkih sustava vanjskih partnera (npr. banaka). Aplikacija bi korisniku nudila opciju unosa gotovinskih transakcija i praćenja mjesečnog priljeva i odljeva novca. Kroz mogućnosti povezivanja kriptovalutnih novčanika i računa s dionicama, korisniku bi se omogućio vremenski pregled vrijednosti njegovih ulaganja korištenjem grafikona. Korisnik bi mogao definirati budžete za zadana vremenska razdoblja koji bi uzimali u obzir trenutno stanje računa i mjesečni priljev i odljev. Aplikacija korisnika obavještava o nadolazećim troškovima, kao i upozorenjima kada se bliži „probijanju“ postavljenih budžeta. Kao dodatnu mogućnost, sustav bi posjedovao popis lokacija bankomata popularnih banaka te bi se korisniku nudila opcija pronalaska obližnjeg bankomata svoje banke. Aplikacija nije zamišljena kao alat za izvršavanje transakcija, već bi služila isključivo kao agregator korisnikove financijske aktivnosti. Originalna zamisao je da aplikacija bude besplatna, ali postoje daljnji koraci i mogućnosti kroz koje bi se aplikacija mogla i monetizirati.

3. Arhitektura sustava

U ovome će se poglavlju pružiti uvid u tehnologije korištene prilikom implementacije programskog rješenja, spomenut će se razvojni alati uz pomoć kojih je rješenje stvoreno te detaljno prikazati arhitektura softverskog rješenja. Za implementaciju ovog rješenja korištena je troslojna klijent-server arhitektura. Mobilna aplikacija predstavlja prezentacijski sloj, REST API služi kao posrednik između vanjskih servisa i baze podataka te tako predstavlja sloj poslovne logike i finalni sloj podatkovne logike predstavljen bazom podataka.



Slika 3.1 Arhitektura sustava Budge

3.1. Korištene tehnologije

Polje računarstva nudi brojne tehnologije i paradigme izrade programskih i informacijskih sustava pa broj programskih okvira za implementaciju raste iz godine u godinu. Lako je izgubiti se u velikom broju tehnologija i principa, ali uz tako široku ponudu jednostavno se

mogu pronaći tehnologije koje pokrivaju točne karakteristike potrebne za projekt koji se izvodi. Prilikom izrade ovog rada nastojalo se u podjednakoj mjeri istražiti nove tehnologije, pritom koristeći programske jezike koji su korišteni tijekom studija. Kroz idućih nekoliko potpoglavlja napravit će se pregled tehnologija korištenih na svakom od slojeva arhitekture.

3.1.1. Baza podatka - PostgreSQL

Za izradu baze podataka izabrana je tehnologija PostgreSQL. PostgreSQL ili Postgres (originalni naziv) je besplatan sustav za održavanje relacijskih baza podataka. Prva verzija izašla je 1996., a od tada je sustav nadograđen mnogim značajkama. Glavni razlog razvoja Postgresa je bila želja njegovih kreatora da izrade bazu podataka koja će podržavati veći broj podatkovnih tipova i u potpunosti biti prilagođena za konkurentnost. Postgres problem konkurentnosti rješava na način da svaka transakcija, koja se odvija na bazi, ima zaseban uvid u stanje baze, a istovremeno se čuva više vremenskih stanja baze. Kao što iz imena PostgreSQL možemo zaključiti, baze podataka u Postgresu razvijaju se koristeći jezik SQL. Neki od značajnijih korisnika ove relacijske baze su: Apple, IMDB, Instagram, Reddit, Spotify itd.

Za razvoj baze potreban je program za pregled i održavanje baza podataka. Na ovome projektu korišten je alat DBeaver. DBeaver je besplatan SQL klijent i alat za administriranje baza podataka. Njegova je prva verzija izašla 2010. godine pa je sam projekt započeo kao hobi projekt programera Sergea Ridera. DBeaver podržava razvoj i SQL i NoSQL baza podataka. Na razvoju projekta Budge, DBeaver je korišten za izvršavanje SQL skripta potrebnih za kreiranje baze podataka i njezinih tablica.

3.1.2. REST servis – Ktor

Za izradu REST klijenta za aplikaciju Budge, korištena je tehnologija Ktor. Ktor je programski okvir za razvoj asinkronih klijenata i servera u programskom jeziku Kotlin. Ktor je tehnologija koju razvija češka tvrtka JetBrains, inače poznata po tome što je razvila i jezik Kotlin te niz iznimno popularnih razvojnih okruženja za raznovrstan popis programskih jezika i tehnologija. Prva verzija Ktora izašla je 2018. godine. Značajka Ktora vrijedna hvale jest što omogućava iznimno brz razvoj asinkronih servera. K tomu, mogućnosti Ktora modularno su razdijeljene u manje cjeline kroz pakete koje je moguće

uključiti koristeći sustav za izgradnju projekta Gradle. Ovisno o potrebama, nužno je samo navesti naziv željenog Ktorova paketa u datoteci za konfiguraciju, a nakon sinkronizacije projekta, server se obogaćuje dodatnim mogućnostima. Postoje paketi za sljedeće funkcionalnosti: rutiranje, stvaranje lokacija, autorizaciju korisnika, serijalizaciju podataka, stvaranje zapisnika aktivnosti, podršku za web-sockete itd.

Za razvoj REST servisa uz pomoć tehnologije Ktor korišteno je razvojno okruženje IntelliJ iDEA Ultimate. IntelliJ je upravo jedan od alata koji tvrtka JetBrains razvija i jedno je od najpopularnijih okruženja za razvoj u programskim jezicima Java i Kotlin. Ovo razvojno okruženje omogućava ubrzan i iznimno ugodan razvoj servisa s obzirom na to da nudi vrlo moćne opcije za otklanjanje pogrešaka i sustav za prediktivno generiranje isječaka koda.

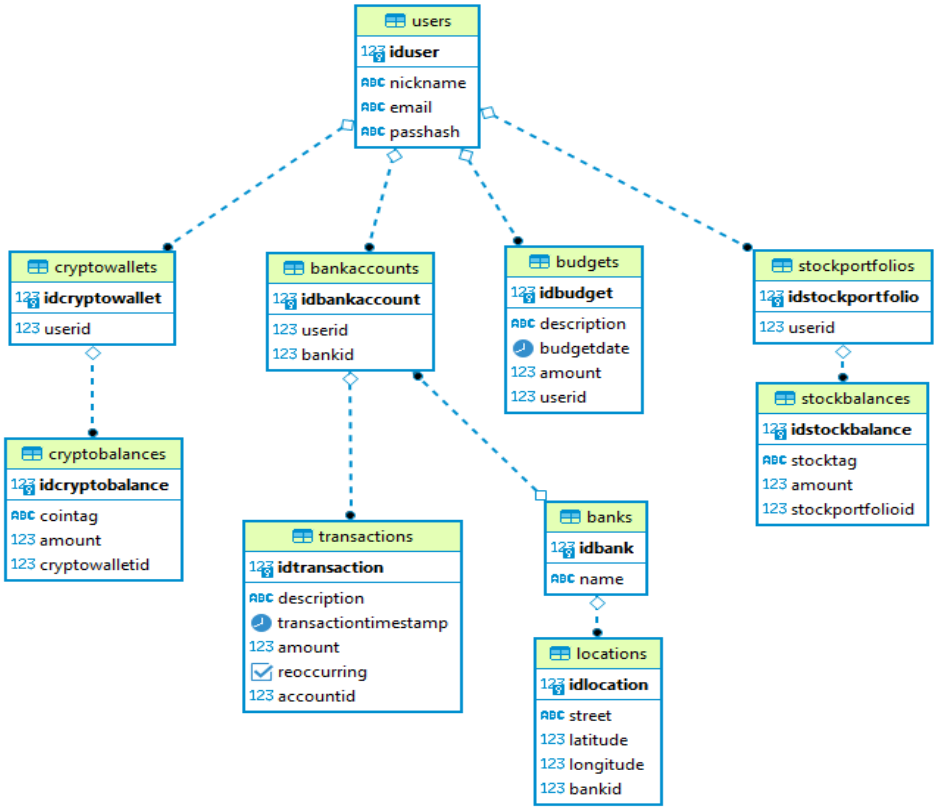
3.1.3. Mobilna aplikacija – Jetpack Compose

Glavni proizvod ovog projekta je mobilna aplikacija za pametne telefone pogonjene operativnim sustavom android. Za razvoj korisničkog sučelja u androidu korištena je tehnologija Jetpack Compose. Jetpack Compose je relativno mlada tehnologija koja je namijenjena da zamijeni stariji sustav izrade korisničkih sučelja koristeći XML datoteke. Compose razvojnom inženjeru omogućava stvaranje korisničkih sučelja na deklarativan način, pišući funkcije u programskom jeziku Kotlin, koje je potom potrebno samo anotirati anotacijom „Composable“. Razvoj android aplikacija i inače se obavlja koristeći programski jezik Kotlin tako da je Compose pomogao u tome da se u što većoj mjeri android razvoj unificira pod jednim jezikom. Kao i kod Ktora, dostupan je sustav za izgradnju projekata Gradle koji omogućava jednostavno dodavanje vanjskih programskih biblioteka, kao i uključivanje vlastitih modula unutar projekta. Ova je značajka iznimno bitna kao što će u nastavku rada biti vidljivo prilikom predstavljanja arhitekture aplikacije.

Za razvoj aplikacije korišteno je razvojno okruženje Android Studio koje se i inače nudi kao oficijelno preporučeni alat. Android Studio je razvojno okruženje koje je osmislila tvrtka JetBrains uz službenu podršku tvrtke Google, a pod čijim je okriljem android ekosustav. Android Studio je okruženje u potpunosti prilagođeno razvoju mobilnih aplikacija pa sadrži napredne značajke poput emulatora android uređaja, mogućnosti povezivanja fizičkih uređaja za direktno izvršavanje koda na njima i mnoge druge. Za potrebe Budge projekta najviše je korišten emulator uređaja na kojem je bilo potrebno podesiti mrežne postavke kako bi se uspješno uspostavila komunikacija s REST servisom.

3.2. Sloj podatkovne logike

Kao što je već spomenuto, sloj podatkovne logike u projektu Budge pokriva baza podataka razvijena koristeći tehnologiju Postgres. Baza je stvorena s primarnim ciljem čuvanja podataka o korisnicima i njihovim budžetima. Tablica s korisničkim podacima sadrži osnovne informacije o korisniku, kao i autorizacijske podatke, poput njegove e-mail adrese i lozinke. Zbog sigurnosnih principa, u bazi se ne čuva lozinka u sirovom tekstualnom formatu, već njezin izračunati *hash*. U produkcijskoj okolini Budge sustav ne bi pohranjivao podatke o korisnikovim transakcijama i stanju portfolija i kriptonovčanika, već bi se ove informacije dohvaćale od njihovih izvornika (banaka i mjenjačnica). Za potrebe izrade demonstracijske verzije aplikacije, stvorene su tablice koje sadrže bazične testne podatke o korisnikovoj imovini. Tablica s korisničkim bankovnim računima sadrži minimalne informacije o korisniku i povezana je s bankom u kojoj je račun otvoren. Stvaranjem 1:n veze između korisnika i bankovnog računa, osigurano je da jedan bankovni račun bude vezan uz isključivo jednog korisnika. Uz informacije o bankama vezana je i tablica koja pohranjuje lokacije bankomata određene banke. Za bankomate u bazi, čuvaju se informacije o nazivu ulice u kojoj se nalaze i koordinate geografske širine i duljine na kojima je bankomat stacioniran. Ove su koordinate neophodne za prikaz lokacija bankomata na karti u aplikaciji. Demonstracijske tablice za informacije o dionicama i kriptovalutama sadrže osnovne informacije, poput oznake i količine dionica/kriptonovčića koje posjeduje korisnik. Tablica za transakcije ujedno služi i kao lokacija za pohranu gotovinskih transakcija koje korisnik unosi te služi i za pohranu ponavljajućih mjesečnih priljeva i odljeva novca. Tablica za budžete pohranjuje podatke o budžetima koje je zacrtao korisnik. Čuvaju se informacije poput opisa potrebe budžeta, datuma za koji je budžet zacrtan te identifikator korisnika koji je dani budžet kreirao. U poveznici s drugim budžetima i ponavljajućim transakcijama na bankovnom računu povezanog korisnika rade se kalkulacije projekcija sačuvanih budžeta.



Slika 3.2 Dijagram baze podataka

3.3. Sloj poslovne logike

Sloj poslovne logike predstavlja centralno mjesto u aplikaciji za obradu i pripremu podataka. U slučaju ovog projekta, REST API izveden tehnologijom Ktor, predstavlja sloj poslovne logike. Kao što je već spomenuto, Ktor od razvojnog inženjera očekuje odabir modula kojima se želi služiti. Za potrebe primanja HTTP upita i slanja odgovora, neophodan je modul za izradu ruta. Dodatne informacije o odabranim modulima i proces njihova uključivanja istražit će se u sljedećem poglavlju. U Budge projektu glavna zadaća API-ja je prosljeđivanje komunikacije između komponenti strukture. API od aplikacije zaprima zahtjeve za podacima iz baze podataka ili od vanjskih API servisa, zatim te zahtjeve prosljeđuje, zaprima odgovore, dohvaćene podatke prepakira u definirane modele i na kraju vraća odgovor aplikaciji. API je odgovoran za : registraciju korisničkih računa, autorizaciju korisnika, dohvaćanje lokacija bankomata, čitanje i pisanje transakcija i budžeta, dohvaćanje trenutnih cijena kriptovaluta i dionica s vanjskih servisa. Korištenjem sigurnosnih ključeva Budge, API je integriran sa dva vanjska API-ja pronađena preko

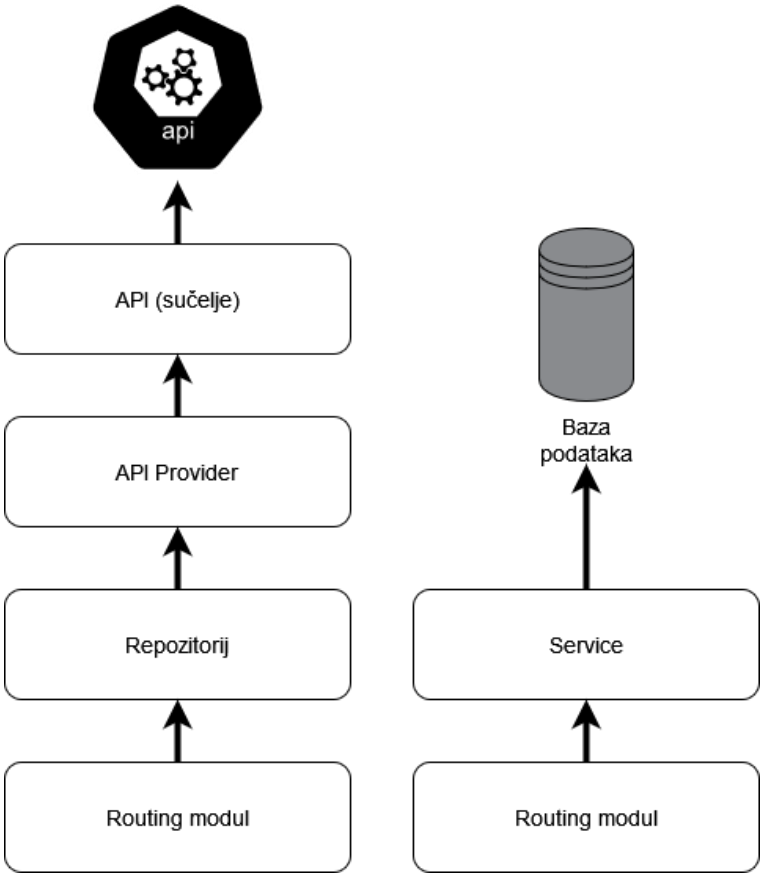
RapidAPI servisa. Coinranking API je javno dostupan servis s ažurnim i detaljnim informacijama o kretanju cijena na tržištu kriptovaluta. Unutar projekta koristi se za dohvaćanje, kako trenutne tako i povijesne cijene zadane mu kriptovalute. Twelve Data je drugi javni API, koji unutar projekta koristimo na sličan način, samo za dohvaćanje informacija o dionicama. Twelve Data u svome odgovoru vraća veći broj informacija nego što je trenutno potreban Budge aplikaciji tako da korištenjem ovih podataka postoji nekoliko mogućnosti proširenja funkcionalnosti aplikacije. U trenutnoj izvedbi jedino se uzima jučerašnja cijena dionice iz vremena zatvaranja tržišta za taj dan. Iz sigurnosnog aspekta, Budge API je zadužen za autorizaciju korisnika koristeći JWT token koji se generira prilikom logina korisnika.

Prilikom izrade servisa korišten je repozitorijski oblikovni obrazac. Tipičan tok programa prilikom dohvaćanja podataka s vanjskih servisa je sljedeći:

- aplikacija šalje zahtjev na jednu od otvorenih ruta unutar modula
- API iz zahtjeva vadi tražene parametre te ih prosljeđuje repozitoriju
- repozitorij, koristeći API provider, dobiva instancu sučelja za komunikaciju s vanjskim API-jem
- repozitorij poziva metodu za dohvat podataka s interneta, prosljeđujući zatražene parametre
- API sučelje prima odgovor te ga vraća repozitoriju
- repozitorij obrađuje zaprimljene podatke te ih iz oblika odgovora prepakira u neku od lokalnih podatkovnih struktura i priprema je za odgovor putem rute
- *routing* modul, ovisno o tome jesu li podaci u željenom obliku, vraća ili strukturu s podacima ili obavijest o grešci klijentu

Komunikacija servisa i baze podataka složena je na nešto drugačiji način. Za pristup bazi podataka korištena je JetBrainsova vlastita biblioteka za pristup bazama Exposed. Exposed funkcionira na način vrlo sličan ORM sustavima stoga, umjesto pozivanja procedura iz baze, kod za manipulaciju tablicama piše se direktno u Kotlinu. Više o ovome u poglavlju o implementaciji. Metode za komunikaciju s bazom izdvojene su u servise (engl. Service) stoga je i tok programa blago drugačiji nego kod vanjskih servisa. Umjesto poziva iz repozitorija, *routing* modul će direktno pozvati metodu iz servisa koji izvršava transakciju

nad bazom podataka te vraćene podatke restrukturira u lokalne modele. Restrukturirani podaci potom se pakiraju u odgovor i vraćaju klijentskoj aplikaciji

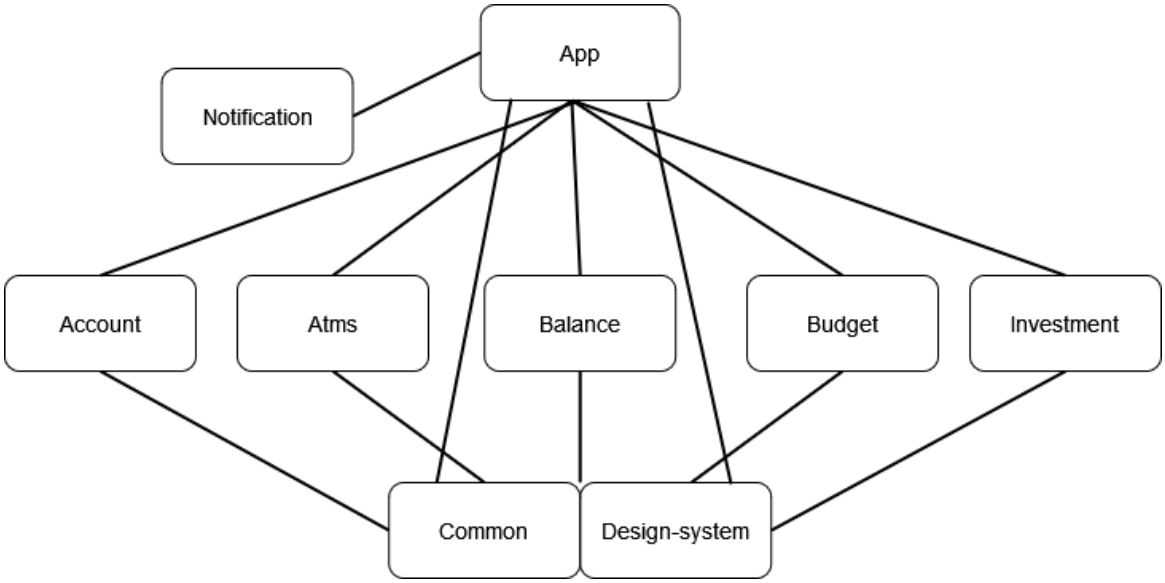


Slika 3.3 Arhitektura REST API-a

3.4. Mobilna aplikacija

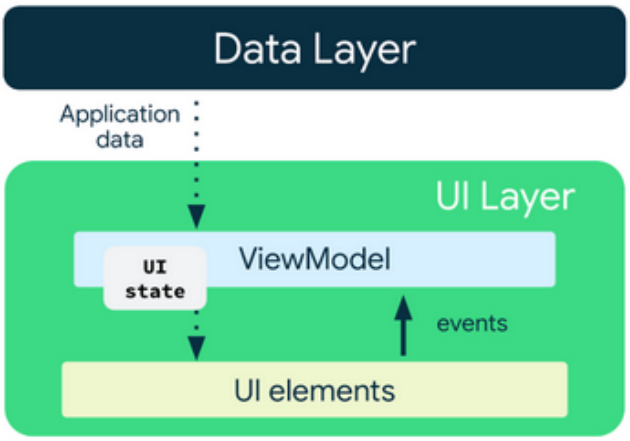
Mobilna aplikacija predstavlja prezentacijski sloj unutar projekta Budge. Prilikom izrade aplikacije nastojalo se koristiti najmodernijim arhitekturalnim pristupima. Primjer ovoga je činjenica da je aplikacija razdijeljena na tzv. *feature module*. Pod ovim se podrazumijeva da se svaka od funkcionalnosti aplikacije odvoji u zaseban programski modul. Samim time i svi programski paketi, koji sadrže modele i logiku povezane uz danu funkcionalnost, odvojeni su od ostatka aplikacije. Ovakvim principom programiranja nastoji se što više poštovati princip razdvajanja odgovornosti (engl. separation of concerns). Jedan od benefita ovakve organizacije aplikacije je znatno brže vrijeme izgradnje projekta prilikom razvoja. Razlog tomu je činjenica da se prilikom izgradnje projekta jedino ponovno

izgrađuju moduli koji su modificirani nakon posljednje izgradnje. Ovakav pristup također omogućava jednostavnu izradu različitih varijanti aplikacije koje nužno ne sadrže iste funkcionalnosti. Pošto je svaka funkcionalnost izdvojena u zaseban modul, moguće ih je modularno uključivati prema potrebi. Jedino se neki resursi, koji se često pojavljuju u više modula, izdvajaju u zajednički modul. Vrlo često koristi se i jedan *design-system* modul koji sadrži osnovnu temu, boje, oblike i zajedničke kontrole aplikacije.



Slika 3.4 Arhitektura modula mobilne aplikacije

Potreba za oblikovanjem arhitekture aplikacije ne staje na modulima. Unutar modula, kod je organiziran poštujući MVVM arhitektonski obrazac. Ovaj obrazac zahtijeva razdvajanje korisničkog sučelja (View) od samih struktura podataka (Model) te odvajanje sve poslovne logike i dohvaćanje podataka u apstraktnu jedinicu zvanu ViewModel. U praksi, korištenjem ovog obrasca poštuje se princip razdvajanja odgovornosti tako što dijelovi korisničkog sučelja jedino komuniciraju s ViewModelom, koji korištenjem repozitorija dohvaća i obrađuje podatke koje zatim servira natrag sučelju. Jedna je od velikih pogodnosti korištenja ovakve arhitekture jednostavnost implementacije asinkronih operacija u aplikaciji.



Slika 3.5 Dijagram MVVM arhitekture

Izvor: Android dokumentacija [3]

Unutar svakog funkcionalnog modula definira se Screen klasa (View), njezin odgovarajući ViewModel te enumeracijska klasa UiState. Unutar UiState enumeracije definira se nekolicina stanja u kojima se može naći korisničko sučelje za određenu funkcionalnost (npr. stanja učitavanja, stanje greške ili stanje uspješnog dohvata podataka). Unutar ViewModela definira se javna varijabla koja sadrži stanje ekrana, a ona se nadzire iz Screen klase pa sučelje reagira na njezine promjene. Tijek asinkronih operacija u ovakvoj arhitekturi bio bi:

- korisničkom interakcijom sa sučeljem poziva se metoda iz ViewModela
- ViewModel putem javne varijable objavljuje stanje učitavanja i asinkrono pokreće dohvaćanje podataka
- Screen reagira na promjenu stanja i pokreće animaciju učitavanja
- ViewModel zaprima rezultat asinkrone akcije i provjerava uspjeh
- ovisno o uspjehu dohvata podataka, objavljuje ili stanje uspjeha ili stanje pogreške
- Screen reagira na promjenu stanja te ili prikazuje dohvaćene podatke ili korisniku prikazuje obavijest o pogrešci.

Kao što je vidljivo, ovaj pristup omogućava iznimno lagano pozivanje asinkronih operacija za dohvat podataka, bez brige o konstantnom osvježavanju korisničkog sučelja.

4. Implementacija programskog rješenja

U ovom će se poglavlju proučiti kako je planirana arhitektura iz prošlog poglavlja implementirana u stvarnim uvjetima. Kroz arhitekturu se prolazi sloj po sloj, usput pregledavajući izazove i metode implementacije prethodno opisanih arhitekturalnih obrazaca i principa.

4.1. Izrada poslužiteljskog dijela

Prvi sloj, koji će se detaljnije secirati, jest poslužiteljski. On uključuje bazu podataka koja pohranjuje već opisane podatke potrebne za rad sustava te REST API servis koji kao posrednik dohvaća podatke s vanjskih servisa ili baze podataka te ih organizira i prosljeđuje klijentskoj aplikaciji.

4.1.1. Baza podataka

Kao što je prethodno spomenuto, za projekt Budge korištena je PostgreSQL relacijska baza podataka. Glavna je zadaća baze u ovom projektu dugoročna pohrana podataka o korisniku, njegovim budžetima i gotovinskim transakcijama te testnih podataka za demonstraciju ideje projekta.

Za izradu baze podataka korišten je spomenuti administracijski alat DBeaver. Unutar DBeavera pisane su skripte za kreiranje baze podataka, kreiranje tablica i njihovih međusobnih relacija. Za ovaj projekt nisu pisane procedure na nivou baze podataka jer je na REST servisu korištena biblioteka Exposed koja stvara direktnu komunikaciju s bazom, tablice unutar baze omotava laganim ORM sustavom te razvojnom inženjeru omogućava modificiranje baze podataka bez pisanja SQL koda. U pozadini Exposed generira SQL skripte i komunikaciju s bazom osigurava koristeći transakcije

Baza za ovaj projekt pokrenuta je lokalno na računalu korištenom za razvoj projekta. Instalacijske datoteke za operacijski sustav Microsoft Windows preuzete su s Postgresove službene stranice. Preuzeta je verzija 15.1 pa je na lokalnom računalu pokrenut Postgres

server. Na serveru je otvoren novi korisnički račun za pristup bazi koji će se koristiti na REST servisu za pristup bazi podataka.

Na serveru je, koristeći korisničko sučelje Dbeavera, kreirana baza Budge za potrebe projekta. Prva i najneophodnija tablica koja je kreirana ona je za pohranu podataka o korisnicima sustava. Sve su tablice kreirane koristeći SQL naredbe zapisane u skriptu.

```
CREATE TABLE users(
idUser INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
nickname VARCHAR(50),
email VARCHAR(50),
passHash VARCHAR(200)
)
```

Kod 4.1 SQL kod za kreiranje tablice korisnika

U kodu za generiranje tablice korisnika može se primijetiti nekoliko iskorištenih mogućnosti SQL baziranih baza. Atribut idUser definiran je kao primarni ključ, što označava da se unutar logike baze on smatra kao polje kojim se identificiraju individualni redovi podataka unutar tablice. Također, primarni ključ koristit će se kao poveznica s drugim povezanim tablicama. Važno je spomenuti i ključnu riječ `IDENTITY` koja u SQL-u označava činjenicu da se vrijednost polja idUser automatski generira prilikom dodavanja retka u tablicu te će baza sama paziti da se koristi unikatna vrijednost. Najčešće `IDENTITY` vrijednosti generiraju se od broja 1 naviše. Važno je što se unutar baze ne čuva stvarna vrijednost korisnikove lozinke, a to bi u slučaju proboja podataka moglo kompromitirati korisnikovu sigurnost, pa se čuva izračunata vrijednost *hasha* njegove lozinke.

```
CREATE TABLE bankaccounts(
idBankAccount INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
userId INT,
bankId INT,
CONSTRAINT fk_user
FOREIGN KEY(userId)
REFERENCES users(idUser),
CONSTRAINT fk_bank
FOREIGN KEY(bankId
```

```
REFERENCES banks(idBank))
```

Kod 4.2 SQL kod za generiranje tablice bankovnih računa

Na primjeru koda za generiranje tablice bankovnih računa moguće je promotriti još jednu od značajki SQL baza. Valja obratiti pozornost na ključne riječi `CONSTRAINT` i `FOREIGN KEY`. Ključna riječ `CONSTRAINT` označava da se na određenu tablicu postavlja ograničenje, ovisno o podatcima koji se u nju smiju unijeti. Ova ograničenja, primjerice, omogućavaju da brojne vrijednosti, koje se unose, budu unutar zadanog okvira, ali mogu zahtijevati da vrijednosti, koje se unose, ne smiju biti prazne i ujedno ograničiti da vrijednosti budu unikatne. U ovome slučaju, koristi se ograničenje `FOREIGN KEY` koje stvara relaciju između tablica na osnovi dvaju povezanih ključeva. Ova tablica posjeduje dva takva ograničenja stoga je povezana s tablicama „users“ i tablicom „banks“. Uzmimo za primjer relaciju s tablicom „users“. Unutar tablice „bankaccounts“ definirana je vrijednost „userId“ koja se ključnom riječju `REFERENCES` povezuje s tablicom „users“. Unutar zagrade, iza naziva tablice, navodi se naziv polja s kojim se povezuje uzeta vrijednost. U ovom slučaju to je polje „idUser“ koje je već spomenuto jer predstavlja primarni ključ i identifikator tablice „users“. Ovakav tip relacije predstavlja vezu 1 : n, što znači da jedan korisnik teoretski može imati više bankovnih računa jer unutar svoje tablice nema čvrstu vezu s tablicom bankovnih računa, dok bankovni račun može imati samo jednog vlasnika jer je njegov identifikator unesen unutar tablice s bankovnim računima.

Prilikom izrade baze pojavio se zanimljiv izazov vezan uz pohranu podataka o vremenu. Pohrana vremenskih informacija općenito se smatra jednom od izazovnijih tema u računarstvu. Vrijeme je iznimno važno, primjerice, za sinkronizaciju servera stoga i najsitnija odstupanja mogu rezultirati katastrofalnim pogreškama u informacijskim sustavima. Najčešći format spremanja vremena je format epohe (engl. Epoch). Epoha predstavlja decimalni broj sekundi koje su prošle od 1. siječnja 1970. godine u 00:00:00. Ovo se naziva epohom UNIX sistema, dok, primjerice, Windows sustav svoju epohu mjeri od 1. siječnja 1601. godine[4]. U ovome slučaju konkretan problem je bio kako sačuvati vrijeme iz Kotlinova objektnog formata LocalDateTime na Ktor servisu unutar Postgres baze. Također, problematičan je bio način slanja istih vremenskih informacija unutar odgovora aplikaciji. Rješenje se pronašlo u Postgresovom podatkovnom formatu `TIMESTAMP WITHOUT TIME ZONE`. Ključna riječ `TIMESTAMP` upravo označava format za pohranu epohe unutar Postgres baze, dok riječi `WITHOUT TIME ZONE`

označavaju korištenje formata bez dodatnih znamenki koje označavaju vremensku zonu u kojoj je zapis kreiran.

Posljednji značajniji detalj implementacije baze podataka je tablica za pohranu lokacije bankomata.

```
CREATE TABLE locations(
idLocation INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
street VARCHAR(50),
latitude DECIMAL,
longitude DECIMAL,
bankId INT,
CONSTRAINT fk_bank
      FOREIGN KEY(bankId)
        REFERENCES banks(idBank)
)
INSERT INTO locations (street, latitude, longitude, bankId)
VALUES ('Zagrebačka', 45.805730507185764, 15.92167354526115,
1),
('šibenska', 45.80539143865583, 15.92876281456953, 1),
('Brače Domany', 45.78972157358379, 15.938142677852525, 1),
('Ozaljska', 45.79747868499665, 15.937845234847167, 1),
...
```

Kod 4.3 SQL kod za kreiranje i popunjavanje tablice s lokacijama bankomata

Kao što je prikazano u isječku koda, ova tablica ima kreiranu referencu prema tablici s bankama. Na ovaj način može se dohvatiti i prikazati isključivo bankomate tražene banke. Same lokacije bankomata čuvaju se u obliku njihovih koordinata, tj. geografske širine i duljine. Ove vrijednosti spremljene su u decimalnom formatu do 15 znamenki iza decimalnog zareza u svrhu što preciznijeg prikazivanja lokacije na karti u aplikaciji. Lokacije bankomata i nazivi banaka jedini su podaci koji su konstante u sustavu Budge te se zapisuju u bazu prilikom njezina stvaranja. Za početnu demo-verziju sustava u obzir su uzete tri popularne hrvatske banke: Privredna banka Zagreb, Zagrebačka banka i Hrvatska poštanska banka. Za potrebe demonstracije prikazane su samo lokacije bankomata na užem području Grada Zagreba. Sve geografske informacije (koordinate bankomata) prikupljene su iz javno dostupnog izvora Google Maps.

4.1.2. Izrada REST API-ja

U ranijim poglavljima već je objašnjena i arhitektura i zamisao korištenja REST API servisa unutar podatkovnog sloja Budge sustava. Dakle, za izradu servisa korištena je tehnologija Ktor koja je omogućila brzi razvoj REST servisa u programskom jeziku Kotlin. Razlog zašto je ciljano na korištenje Ktora je nastojanje da većina programskog koda cijelog Budge projekta (i klijentskog i poslovnog sloja) bude napisana u jednom jeziku: Kotlinu. Na ovaj način promjene konteksta tijekom razvoja su puno blaže i jednostavnije je prenijeti potrebne promjene s jednog sloja na drugi. Korištenje srodnih tehnologija omogućilo je korištenje istih biblioteka na različitim slojevima kako bi se određeni dijelovi koda što više ujednačili (npr. pozivi prema vanjskim servisima). Također, korištenje Kotlina na više slojeva omogućilo je korištenje istih modela i istih biblioteka za serijalizaciju podataka unutar servisa i aplikacije.

Kao što je spomenuto, Ktor je građen kao sustav za modularno slaganje asinkronih web-servisa. Moduli se uključuju na isti način na koji instaliramo vanjske programske biblioteke. Potrebno je jedino navesti nazive serverskih modula koji se žele koristiti unutar konfiguracijske datoteke sustava za vođenje ovisnosti Gradle. Potom svaki od modula generira statičku klasu za inicijalizaciju parametara. Inicijalizacija se obavlja prilikom pokretanja serverske aplikacije.

```
implementation("io.ktor:ktor-server-core-jvm:$ktor_version")

implementation("io.ktor:ktor-server-content-negotiation-
jvm:$ktor_version")

implementation("io.ktor:ktor-serialization-gson-
jvm:$ktor_version")

implementation("io.ktor:ktor-server-auth-jwt-
jvm:$ktor_version")

implementation("io.ktor:ktor-server-locations-
jvm:$ktor_version")

implementation("io.ktor:ktor-server-netty-jvm:$ktor_version")
```

Kod 4.4 Primjer uključivanja serverskih modula

Promotrimo isječak koda 4.4. Ovaj isječak prikazuje serverske module uključene u projektu Budge, izvučene iz konfiguracijske datoteke „build.gradle.kts“. Moduli uključeni u projektu obavljaju sljedeće zadaće:

- osnovne komponente za rad servera
- mogućnost primanja različitih formata podataka unutar HTTP zahtjeva
- serijalizacija podataka u format JSON koristeći Googleovu serijalizacijsku biblioteku Gson
- mogućnost autentikacije i autorizacije korisnika korištenjem JWT tokena preko Bearer metode autorizacije
- mogućnost grupiranja servisnih krajnjih točaka (engl. endpoint) pod zajedničke lokacije
- implementacija HTTP servera uz pomoć popularne biblioteke Netty.

Ovi moduli predstavljaju osnovne gradivne komponente koje omogućavaju pravilan rad Budge API-ja. Neki od modula se ne korisne direktno, već ih Ktor interno koristi kako bi inicijalizirao server. Oni moduli koji omogućavaju veću količinu modifikacije zahtijevaju od razvojnog inženjera da ih postavi i inicijalizira.

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0",
module = Application::module)
        .start(wait = true)
}
fun Application.module() {
    configureSerialization()
    configureSecurity()
    configureRouting()
}
```

Kod 4.5 Kod za pokretanje Ktor servera

Kao što je vidljivo iz isječka 4.5., unutar funkcije `Application.module()` potrebno je inicijalizirati i konfigurirati module za serijalizaciju, sigurnost i rutiranje zahtjeva. Glavna funkcija programa instancira objekt servera, koristeći ranije uključenu Netty biblioteku, te ga postavlja da osluškuje na predefiniranom portu 8080 i predefiniranoj adresi. U pozadini predefinirana adresa je 172.0.0.1, tj. popularni „localhost“. Prije

pokretanja samog servera, Ktor poziva funkciju `module()` kako bi server nadogradio dodanim konfiguracijama.

Ktor u svojoj konfiguraciji nudi nekoliko izbora serverske sigurnosti. Moguće je propuštati sve upite bez ikakve sigurnosne provjere, koristiti osnovnu provjeru e-mail adrese i lozinke za svaki upit, implementirati zaštitu preko JWT tokena, sesijsku autorizaciju čuvanjem podataka o korisniku u njegovoj sesiji ili integracijom s LDAP sustavom. Na Budge projektu odlučeno je koristiti JWT metodu autorizacije zbog nekoliko razloga. Implementacija je bila relativno jednostavna pa ova metoda omogućava izdavanje tokena u slučaju zadovoljenja određenih uvjeta (podudaranja e-mail adrese i lozinke). Također, JWT tokeni omogućuju dodatnu razinu sigurnosti time što imaju ograničeno vremensko trajanje. Tokeni nakon definiranog vremena ističu pa onemogućuju napadače da iskoriste stari ukradeni token za komunikaciju sa servisom. JWT autentikacija konfigurirana je ovako:

```
fun Application.configureSecurity() {
    authentication {
        jwt("auth") {
            realm = JwtSecrets.REALM.value
            val audience = JwtSecrets.AUDIENCE.value
            verifier(
                JWT
             .require(Algorithm.HMAC256(JwtSecrets.SECRET.value))
                .withAudience(audience)
                .withIssuer(JwtSecrets.ISSUER.value)
                .build())
            validate { credential ->
                if
(credential.payload.audience.contains(audience))
JWTPrincipal(credential.payload) else null
            }
            challenge { _, _ ->
                call.respond(HttpStatusCode.Unauthorized,
"Token is not valid or has expired")
            }}}}
```

Kod 4.6 Implementacija JWT autentikacije

Unutar autentikacijskog modula definira se nova ruta koja koristi JWT shemu. Za ovu rutu moraju se definirati oblast (engl. realm), publika (engl. audience), tajna (engl. secret) i

izdavač (engl. issuer). Oblast predstavlja opis opsega koji ova autentikacijska ruta otvara, publika predstavlja serverske rute koje će ova autentikacijska ruta štititi. U ovom slučaju publiku predstavljaju sve rute na serveru, uz nekoliko iznimki koje će se kasnije spomenuti. Izdavač predstavlja adresu servera za koji generirani JWT tokeni vrijede pa je tajna ustvari jednostavan niz znakova koji služi za dodatno kompliciranje enkripcije ključa. Sve ove vrijednosti navedene su u Secrets.kt datoteci koja je izdvojena s Git sustava za verzioniranje kako bi se spriječilo curenje vrijednosti. Nakon što su definirani parametri, kojima se JWT tokeni provjeravaju, potrebno je navesti potencijalne rezultate provjere. U slučaju da je JWT token ispravan, zahtjev se propušta do tražene rute, a u slučaju neispravnog tokena klijent se odbija sa statusnom šifrom 401 (nedopušten pristup).

Na Budge serveru definirane su rute za baratanje korisničkim računom (kreiranje, prijava i odjava), dohvaćanje lokacija bankomata, dohvaćanje korisnikovih transakcija, dohvaćanje korisnikovih kriptovaluta i dionica, provjeru cijena dionica i kriptovaluta te baratanje korisnikovim budžetima. Sve rute na serveru zaštićene su već definiranom jwt shemom osim točaka za registraciju i prijavu. Ne bi imalo smisla zatvoriti i ove točke pošto novi i neprijavljeni korisnici nisu imali prilike ranije dobiti JWT token tako da ne bi mogli pristupiti serveru. Proces generiranja JWT tokena i stvaranja korisničkog objekta prilikom prijave predstavljen je u sljedećem isječku koda:

```
token=JWT.create().withAudience(JwtSecrets.AUDIENCE.value)
                        .withIssuer(JwtSecrets.ISSUER.value)
                        .withClaim("username",user!!.nickname)
         .withExpiresAt(Date(System.currentTimeMillis() +
         TimeUnit.MINUTES.toMillis(45)))
        .sign(Algorithm.HMAC256(JwtSecrets.SECRET.value))
                    val response = User(
                        user!!.idUser.value,
                        user!!.nickname,
                        token,
                        bankAccount = account,
                        cryptoWalletId = wallet!!.idWallet,
                stockPortfolioId=portfolio!!.idStockPortfolio)
                        call.respond(response)
```

Kod 4.7 Generiranje JWT tokena i kreiranje User objekta

Vidljivo je da se token generira koristeći iste parametre koji se koriste prilikom provjere autentičnosti tokena. Definira se ista publika i izdavač pa se u token dodaje korisničko ime

uspješno prijavljenog korisnika. U testnoj verziji aplikacije vrijeme trajanja tokena postavljeno je na 45 minuta od trenutka izdavanja, što bi u produkciji bilo smanjeno te bi se token morao češće osvježiti. Token se potom enkriptira s dodatkom tajne. Zatim se uz ostale korisničke podatke dohvaćene iz baze podataka pakira u objekt „user“ i putem serijaliziranog odgovora šalje natrag aplikaciji.

Serijalizacija na Budge serveru namještena je da objekte serijalizira u JSON format, koristeći Googleovu serijalizacijsku biblioteku Gson. Kako bi se klase mogle serijalizirati, potrebno ih je označiti anotacijama koje sadrže podatke o nazivu i redoslijedu serijalizacije u JSON objekte. U programskom jeziku Kotlin ovo je vrlo jednostavno izvesti dodavanjem anotacija na parametre u konstruktoru podatkovnih klasa. Podatkovne klase u Kotlinu predstavljaju brži način deklariranja klasa za koje se automatski generiraju konstruktori, *getteri* i *setteri*. Primjer jedne takve anotirane klase može se vidjeti ovdje:

```
data class AtmLocation (
    @SerializedName("idLocation") val id: Int? = null,
    @SerializedName("street") val street: String? = null,
    @SerializedName("latitude") val latitude: Double? = null,
    @SerializedName("longitude") val longitude: Double? =
null,
    @SerializedName("bankId") val bankId: Int? = null
)
```

Kod 4.8 Primjer podatkovne klase anotirane za serijalizaciju

Kao što je vidljivo iz isječka s klasom za lokacije bankomata, svako od polja unutar klase ima dodanu anotaciju `@SerializedName` i definiranu osnovnu vrijednost. Osnovne vrijednosti potrebne su za slučajeve opcionalnih parametara koji mogu biti zapostavljeni prilikom serijalizacije, ali oni ne limitiraju uporabu ni na koji način. Parametre, koji su potencijalno prazni, moguće je provjeriti klasičnim provjerama ili ključnom riječi `?.let{}` u programskom jeziku Kotlin. Naziv unesen unutar zagrade, iza anotacije, definira kako će biti nazvano polje za dani parametar unutar generiranog JSON objekta.

Još jedan važan detalj implementacije REST API-ja je komunikacija s bazom podataka. Kao što je već spomenuto, za komunikaciju s bazom korištena je biblioteka Exposed. Ova biblioteka preko laganog ORM omotača oko baze podataka omogućava jednostavne

CRUD operacije, koristeći dvije komponente: objekt koji predstavlja tablicu u bazi i klasu koja predstavlja kotlinovsku reprezentaciju redova iz tablice. Primjer ovakvog načina rada s bazom secirat će se u sljedećem isječku klase s podacima o bankama:

```
object Banks : IntIdTable(columnName = "idbank") {
    val name = varchar("name", 50)
}
class DbBank(val idBank: EntityID<Int>) : IntEntity(idBank) {
    companion object : IntEntityClass<DbBank>(Banks)
    var name by Banks.name
}
fun getAllBanks(): List<DbBank> {
    val list = mutableListOf<DbBank>()
    transaction(DatabaseProvider.provideDb()) {
        SchemaUtils.create(Banks)
        list.addAll(DbBank.all().toList())
    }
    return list
}
```

Kod 4.9 Primjer pristupa bazi podataka preko biblioteke Exposed

Prvo se definira objekt `Banks` koji unutar Postgres baze predstavlja tablicu s naznačenim primarnim ključem „idbank“ te jednim tekstualnim parametrom „name“ koji sadrži ime banke. Zatim se kreira klasa `DbBank` koja se putem statičkog objekta povezuje s objektom `Banks`. Ova klasa služi za konverziju podataka iz tipova u kojima ih skladišti Postgres u primitivne tipove iz jezika Kotlin. Jedan primjer korištenja ovih objekata za manipulaciju podacima iz baze podataka je funkcija `getAllBanks()`. Funkcija u sebi kreira praznu listu banaka, a nakon toga započinje transakciju nad bazom preko konekcije koju joj dostavlja `DatabaseProvider`. Zatim, pomoću funkcije `SchemaUtils.create()` provjerava se postoji li navedena tablica ili ju je potrebno nanovo kreirati. Potom statički objekt na klasi `DbBank` omogućava korištenje metode `.all()` koja dohvaća sve redove u bazi koji se zatim dodaju u praznu listu koja se naposljetku vraća kao rezultat funkcije.

4.2. Izrada klijentskog dijela

U ovome poglavlju pobliže će se predstaviti detalji implementacije klijentskog prezentacijskoga sloja projekta Budge. Vidjet će se kako su arhitektonski obrasci, spomenuti u poglavlju o arhitekturi, implementirani u praksi.

4.2.1. Mobilna aplikacija

Mobilna aplikacija Budge razvijena je za uređaje pogonjene operacijskim sustavom android, koristeći Jetpack Compose tehnologiju za izradu korisničkog sučelja. Aplikacija je izgrađena prateći princip jedne aktivnosti, a razdijeljena je na veći broj funkcionalnih modula, kao što je spomenuto u opisu arhitekture. Zajednička točka poveznica cijele aplikacije je ta jedina aktivnost – MainActivity. Aktivnosti u androidu predstavljaju ekrane koji omogućavaju interakciju između korisnika i aplikacije. S vremenom se arhitektura i logika oko aktivnosti u androidu mijenjala pa se u modernijim pristupima teži imati što manji broj ili, idealno, samo jednu aktivnost. Prijelazna metoda implementacije aplikacije s jednom aktivnošću bili su fragmenti, koji su izmjenjive komponente aktivnosti kroz koje se može navigirati koristeći navigacijske grafove.

Dolazak Jetpack Compose tehnologije ponovno je promijenio ovu paradigmu i posebno olakšao izradu aplikacije s jednom aktivnošću ako se razvoj započinje od nule. Osnova Compose okvira su anotirane funkcije koje pozivaju interne funkcije na uređaju za iscrtavanje interaktibilnih komponenti na ekran uređaja. Nova logika izrade aplikacija podrazumijeva izradu posebne *Screen* klase za svaki od funkcionalnih ekrana koji se planiraju koristiti u aplikaciji. Ovi ekrani se zatim ispunjavaju kompozabilnim komponentama koje su sortirane u svojim odvojenim modulima te je svaka zadužena za prikaz drugačijih podataka. Jedina zajednička točka *Screenova* u aplikaciji je MainActivity, aktivnost u kojoj se instancira navigacijski kontroler iz oficijelnog paketa Jetpack Compose Navigation. Implementacija Compose navigacije u Budge aplikaciji prikazana je kroz sljedećih nekoliko isječaka.

```
sealed class Screen(
    val route: String,
    @StringRes val resourceId: Int,
    @DrawableRes val iconId: Int
    ) {
```

```
    object Budget : Screen("budget", R.string.budget,
R.drawable.wallet)
    object Investment : Screen("investment",
R.string.investment, R.drawable.monitoring)
    object Balance : Screen("balance", R.string.balance,
R.drawable.balance)
    object Atms : Screen("atms", R.string.atms,
R.drawable.location_on)
    object Account : Screen("account", R.string.account,
R.drawable.account_circle)
}

val items = listOf(
    Screen.Budget,
    Screen.Investment,
    Screen.Balance,
    Screen.Atms,
    Screen.Account
)
```

Kod 4.10 Deklariranje objekta s resursima ekrana u aplikaciji

Prvi korak u izradi navigacije je definiranje objekta koji će čuvati relevantne podatke za navigacijske rute u aplikaciji. U ovom slučaju ideja je koristiti navigacijsku traku na dnu ekrana, što znači da će za svaki od ekrana trebati definirati ikonicu i naslov koji se prikazuje. Iz tog razloga kreirana je klasa *Screen* koja u sebi čuva podatke o ruti na kojoj se određeni ekran nalazi, identifikator tekstualnog resursa, koji će se ispisati kao oznaka u navigaciji, te identifikator vizualnog resursa, koji će se koristiti kao ikonicu. Nakon toga izrađen je po jedan objekt za svaki od ekrana sa svim odgovarajućim resursima. Definirani objekti skupljeni su u listu items koja će se kasnije iskoristiti.

```
@Composable
fun Navigation(navController: NavHostController) {
    NavigationBar {
        val navBackStackEntry by
navController.currentBackStackEntryAsState()
        val currentDestination =
navBackStackEntry?.destination
        items.forEach { screen ->
            NavigationBarItem(
```

```
                icon = { Icon(painterResource(id =
screen.iconId), contentDescription = null) },
                label = {
Text(stringResource(screen.resourceId)) },
                selected = currentDestination?.hierarchy?.any
{ it.route == screen.route } == true,
                onClick = {
                    navController.navigate(screen.route) {
                        launchSingleTop = true
                        // Restore state when reselecting a
previously selected item
                        restoreState = true
                    }}}}}
```

Kod 4.11 Funkcija za izgradnju navigacijske trake

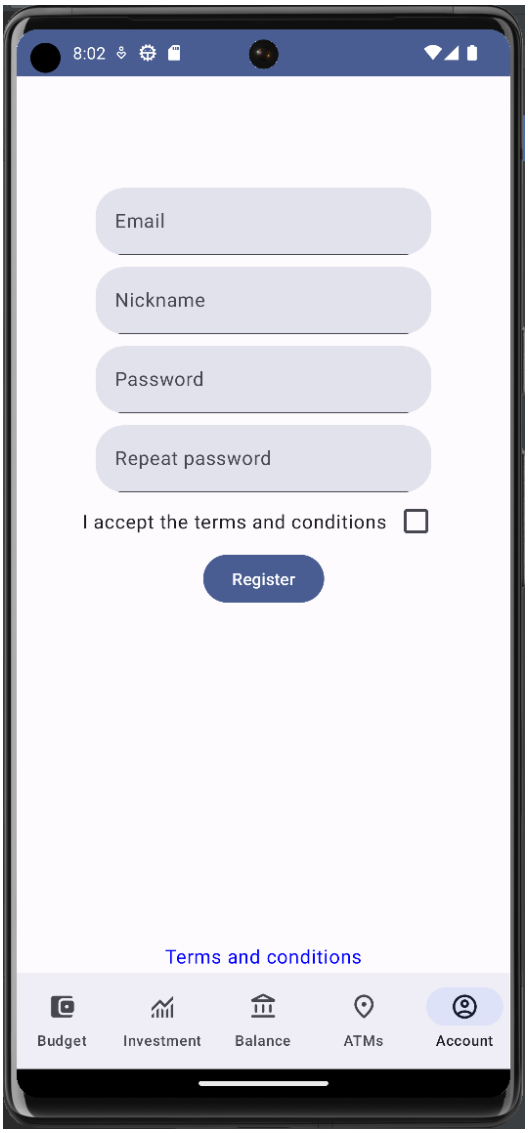
U isječku 4.11. kreirana je kompozabilna funkcija za prikazivanje navigacijske trake. Kao što je vidljivo, ova funkcija za parametar prima navigacijski kontroler kojim će se navigirati kroz ekrane, a trenutno i prošlo stanje navigacije se spremaju zbog navigiranja putem sistemske tipke natrag. Potom za svaki element liste ekrana, koja je stvorena u prošlom koraku, izrađuje se NavigationBarItem koristeći njegove definirane resurse. Uz to, u navigaciji se postavlja drugačiji stil označenom elementu i na korisnikove klikove reagira se tako što se obavlja navigacija na rutu odabranog elementa.

```
val navController = rememberNavController()
          Scaffold(bottomBar =
{Navigation(navController)}) {
      NavHost(navController = navController,
      startDestination = "balance") {
            composable("budget") { BudgetScreen() }
            composable("investment") { InvestmentScreen() }
            composable("balance") { BalanceScreen() }
            composable("atms") { AtmsScreen() }
            composable("account") { AccountScreen() }
      }}
```

Kod 4.12 Povezivanje navigacijskog kontrolera i navigacijske trake

U isječku 4.12. prikazano je krajnje povezivanje navigacijskog kontrolera i Navigation() funkcije kako bi se na ekranu iscrtala funkcionalna navigacija. Navigacija se postavlja kao

bottomBar element okvira Scaffold, što jamči da će navigacija uvijek biti pozicionirana na dnu ekrana.



Slika 4.1 Izgled registracijske forme

Kako bi započeo koristiti aplikaciju, korisnik mora kreirati besplatan račun koristeći registracijsku formu. Od korisnika se traži unos e-mail adrese, korisničkog imena i lozinke koju mora potvrditi. Od korisnika se očekuje unos svih traženih podataka, što se osigurava validacijom forme prije predaje. Osim toga, od korisnika se traži potvrda da prihvaća ugovor i klauzule o baratanju privatnim podacima. Potom se istražuje implementacija MVVM arhitekture koja je spominjana u prošlom poglavlju. Korisnik će klikom na gumb

„Register“ pokrenuti metodu za registraciju unutar ViewModela, što će rezultirati vizualnim promjenama na ekrane i asinkrono pokrenuti komunikaciju s REST API-jem.

```
fun registerAccount() {
        _viewState.postValue(AccountUiState.LOADING)
        val nickname = textFieldValues["Nickname"]!!
        val email = textFieldValues["Email"]!!
        val pass = textFieldValues["Password"]!!
        viewModelScope.launch {
            val result =
AccountRepository.registerAccount(nickname, email, pass)
            if (result != null) {
                _viewState.postValue(AccountUiState.LOGIN)
            }
            _viewState.postValue(AccountUiState.ERROR)
        }
    }
```

Kod 4.13 Metoda za registraciju unutar ViewModela

Kao i u primjeru iz poglavlja o arhitekturi, ViewModel odašilje stanje učitavanja ekranu, što će pokrenuti animaciju učitavanja. Pritom pokreće korutinu na kojoj se obavlja poziv Account repozitoriju koji će kao rezultat akcije vratiti informaciju je li registracija uspjela. Ovisno o rezultatu, prikazuje se ili ekran za prijavu ili informaciju o grešci. Akcija registracije korisnika unutar repozitorija izgleda ovako:

```
suspend fun registerAccount(
        nickname: String,
        email: String,
        pass: String
    ) : String? {
        return AccountApiProvider
            .provideAccountApi()
            .registerAccount(nickname, email, pass)
            .body()
    }
```

Kod 4.14 Metoda za registraciju unutar repozitorija

Unutar repozitorija funkcija registerAccount blokira izvođenje na korutini na kojoj se nalazi (zato je ova funkcija pozvana na novoj korutini). Korištenjem ApiProvidera dobiva instancu API sučelja koje sadrži definirane rute za komunikaciju s API servisom.

ApiProvider je jednostavan singleton objekt koji kreira instancu sučelja po potrebi. Repozitorij sučelju prosljeđuje korisnikove unesene parametre, a u slučaju odgovora ViewModelu vraća tijelo HTTP odgovora.

Za HTTP komunikaciju u aplikaciji korištena je vrlo popularna biblioteka Retrofit. Retrofit, kao što je već objašnjeno, obavlja pozive instanciranjem Retrofit objekta koji za osnovne parametre prima osnovni URL servisa s kojim se namjerava komunicirati i instancu biblioteke za serijalizaciju i deserijalizaciju podataka. U ovome slučaju, koristi se ista serijalizacijska biblioteka kao i na API-ju, Gson. Drugi dio Retrofit poziva je API sučelje koje definira rutu servisa na koji se cilja, sigurnosne tokene za autorizaciju, dodatne parametre zahtjeva i podatkovni tip koji se očekuje kao odgovor servisa. Primjer ovakvog sučelja vidljiv je na objektu za registraciju korisnika:

```
@POST("/register")
    suspend fun registerAccount(
        @Query("nickname") nickname: String,
        @Query("email") email: String,
        @Query("pass") pass: String
    ): Response<String>
```

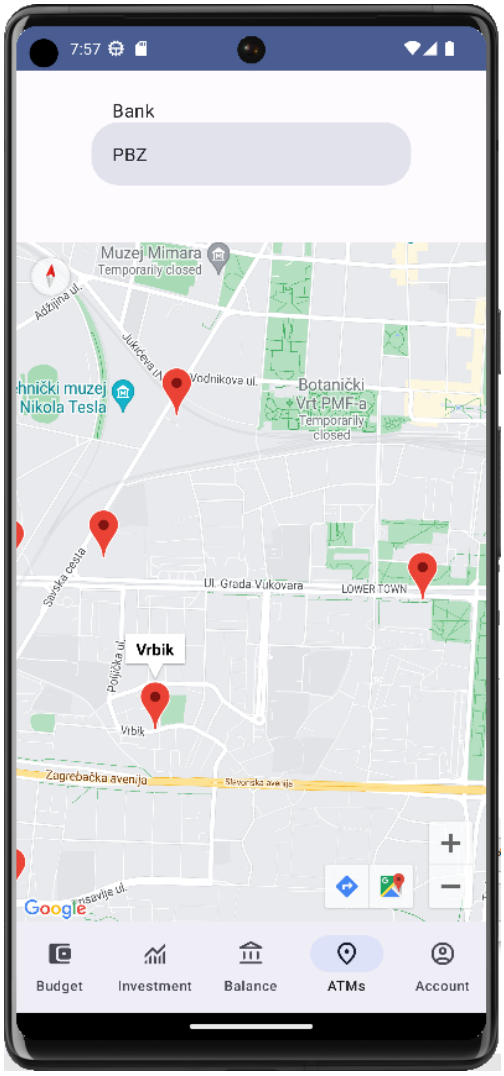
Kod 4.15 Metoda za registraciju unutar API sučelja

Ova metoda cilja rutu „register“ na servisu te kao parametre zahtjeva šalje nadimak, e-mail adresu i lozinku koje je unio korisnik. Za odgovor očekuje se zaprimiti tekstualnu poruku koja nije prazna, kao znak uspješne registracije. Sva komunikacija između aplikacije i REST API-ja je bazirana na ovoj metodologiji stoga svaki od modula ima svoj ekran, ViewModel, repozitorij, ApiProvidera i Api sučelje.

Korisnička prijava funkcionira na isti način kao i registracija. Unosom podataka u formu poziva se metoda u ViewModelu koja preko repozitorija u slučaju uspjeha dobiva objekt korisnika koji sadrži identifikatore njegovih bankovnih, dioničarskih i kriptovalutnih računa. Uz ove informacije, aplikacija ujedno zaprima i generirani JWT token, koji prilaže u zaglavlje zahtjeva, prema svim zaštićenim rutama na API servisu. Uspješna prijava pokreće mehanizam spremanja korisnikovih identifikacijskih informacija u unutarnju memoriju uređaja, što omogućuje automatsku prijavu prilikom paljenja aplikacije. Nakon

korisničke prijave, nudi se opcija odjave koja servisu javlja da je potrebno poništiti izdani JWT token.

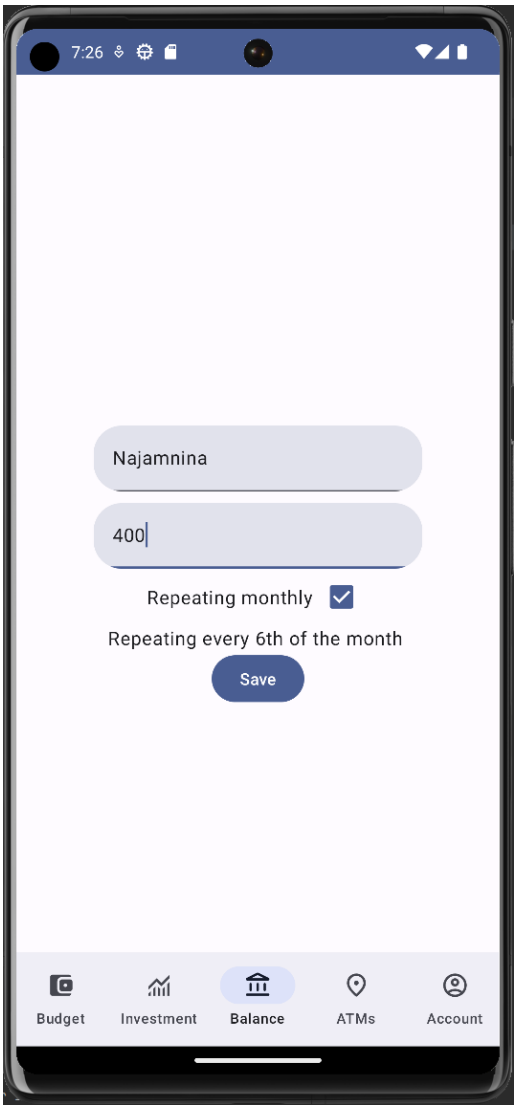
Kada korisnik označi ikonicu „ATMs“, aplikacija navigira na ekran za prikaz lokacija bankomata. U slučaju da korisnik ima registriran bankovni račun, otvaraju mu se lokacije bankomata njegove banke, a u slučaju da ga nema, otvara se prva banka prema abecednom redoslijedu. Kroz padajući meni moguće je odabrati banku čije se lokacije bankomata prikazuju. Na ovome ekranu je integrirana Googleova biblioteka za rad s Google Maps API-jem koja omogućava prikaz lokacija na korisnicima poznatom sučelju popularne aplikacije Google Maps.



Slika 4.2 Izgled ekrana s lokacijama bankomata

Korisniku se na prikazu karte pojavljuju oznake s lokacijama bankomata. Klikom na oznaku korisnik može vidjeti naziv ulice u kojoj se nalazi bankomat.

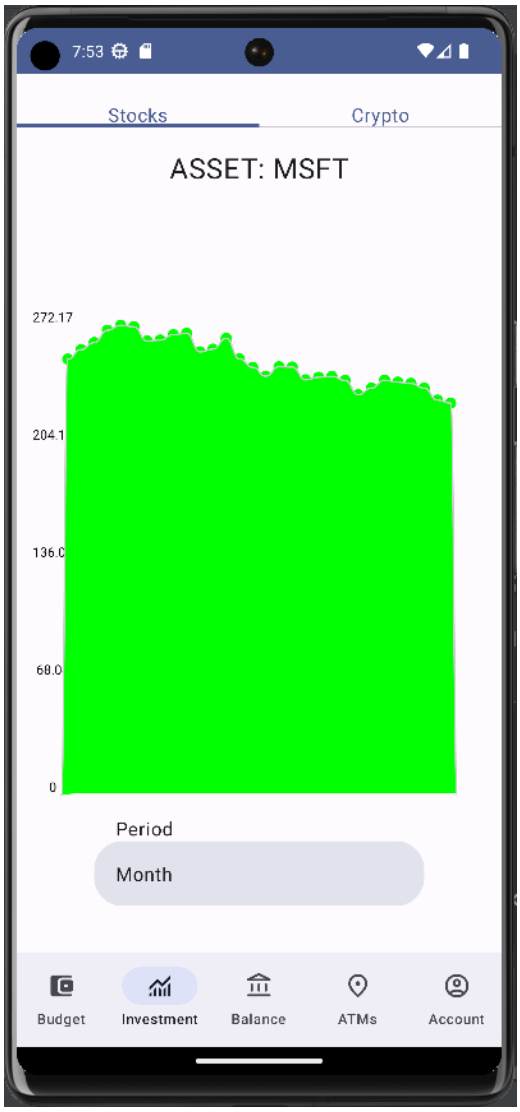
Odabirom ikonice „Balance“, korisniku se otvara ekran s podacima o njegovim transakcijama. U slučaju da korisnik nije povezao svoj bankovni račun, nudi mu se mogućnost povezivanja. U demo-verziji aplikacije korisnik može odabrati banku povezanu uz račun, koji se generira, te se na servisu generira pet nasumičnih transakcija kako bi se simulirala aktivnost na računu. Na ovome ekranu korisnik ima pregled svoje ukupne neto vrijednosti i vidljiva mu je lista svih transakcija s bankovnog računa i ručno unesenih gotovinskih transakcija. Ulaskom na drugi odjeljak ovoga ekrana, korisniku se otvara popis njegovih mjesečnih priljeva i odljeva novaca. Klikom na gumb, na ovome ekranu nudi se opcija unosa gotovinske ili ponavljajuće transakcije.



Slika 4.3 Izgled ekrana za dodavanje transakcije

Korisnik može unijeti opis i iznos transakcije (pozitivan ili negativan) i klikom na polje s kvačicom odabrati želi li transakciju dodati kao jednokratnu ili ponavljajuću. Klikom na polje s datumom, otvara se dijalog za odabir vremena koji korisniku omogućava odabir datuma i vremena gotovinske transakcije ili dana u mjesecu kada se događa ponavljajuća transakcija. Ručno dodane transakcije moguće je mijenjati i brisati.

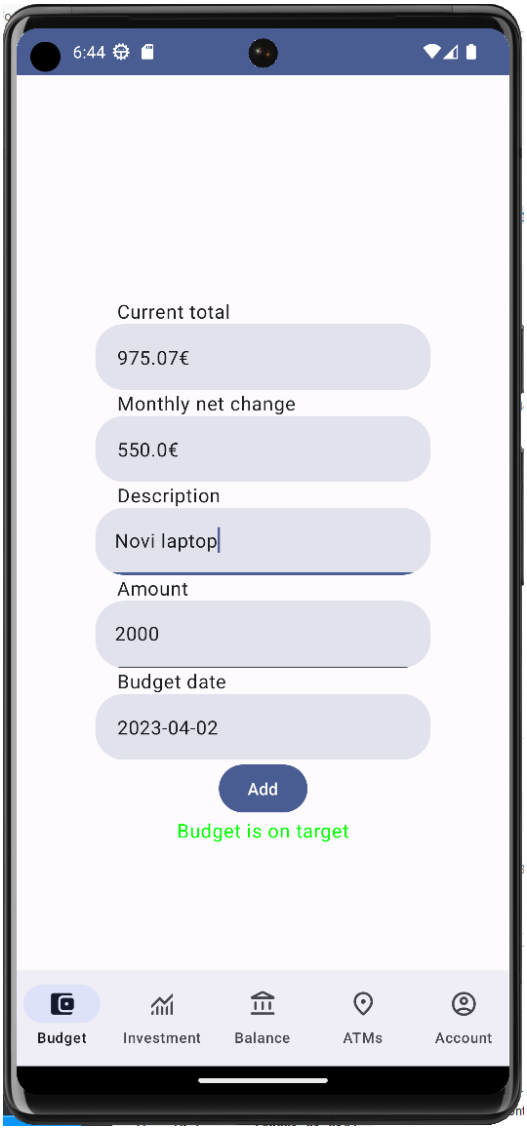
Odabirom ikonice „Investments“, korisniku se otvara prozor s informacijama o njegovim ulaganjima. U slučaju da nema već povezan portfolio s dionicama ili kriptonovčanik, korisniku se nudi opcija da to učini. U demo-verziji aplikacije, kao i kod bankovnih transakcija, na servisu se simulira unos 3 dionice ili kriptovalute na korisnikov račun. Kroz odvojene poglede korisnik može vidjeti količinu imovine (dionice/kriptovalute) koju posjeduje, trenutnu cijenu jedne jedinice i ukupnu vrijednost imovine koju posjeduje. Klikom na jednu od dionica ili kriptovaluta, otvara se ekran s detaljnim pogledom na kretanje cijene imovine. Kroz grafički prikaz moguće je vidjeti kretanje cijene imovine u odabranom razdoblju (dan, tjedan, mjesec).



Slika 4.4 Izgled grafikona kretanja cijene imovine

Odabirom ikonice „Budget“ korisniku se otvara zaslon s mogućnostima budžetiranja. Prikazuje mu se popis zacrtanih budžeta s informacijama poput: opisa, roka, iznosa te indikatora je li projekcija budžeta trenutno pozitivna ili nije. Za kalkulaciju projekcije uzimaju se u obzir drugi budžeti kojima se vremenski rokovi preklapaju, trenutno stanje računa korisnika i sve ponavljajuće transakcije koje je naveo u meniju za stanje računa. Klikom na gumb za dodavanje, otvara se forma za kreiranje novog budžeta. Forma od korisnika zahtijeva ranije spomenute informacije i nudi mu odabir datuma kroz `DatePicker` dijalog. Prilikom kreiranja budžeta isto tako se obavlja kalkulacija projekcije pa se korisnika pomoću obojenog teksta na dnu ekrana obavještava izgleda li

budžet koji kreira s trenutnim stanjem ostvarivo. Kako bi se olakšalo planiranje dostavljaju se informacije o trenutnom totalu na računu, kao i mjesečne projekcije neto promjene stanja.



Slika 4.5 Izgled ekrana za kreiranje budžeta

4.3. Sigurnosne značajke

U ovome poglavlju napravit će se presjek sigurnosnih značajki implementiranog sustava. S obzirom na kritičnost financijskih podataka i razinu pažnje koju svjetska regulacijska tijela tijekom zadnjeg desetljeća promoviraju, nužno je zaključiti da je sigurnost korisničkih podataka u sustavu Budge prioritet.

U produkcijskom okruženju sustav Budge ne bi čuvao nikakve zapise o korisnikovim vezanim financijskim računima. Ne bi se čuvali nikakvi podaci vezani uz korisnikove bankovne transakcije, stanje dioničarskog portfolija ili stanja na kriptonovčaniku. Sve financijske informacije dohvaćale bi se sigurnim i enkriptiranim putem od partnerskih pružatelja usluga (banaka i mjenjačnica). U produkcijskoj bazi informacije o budžetima i gotovinskim transakcijama enkriptirale bi se korištenjem determinističke enkripcije simetričnim ključem koja bi omogućila vezanje blokova podataka s njihovim vlasnikom.

U produkcijskoj i demo-verziji projekta implementirana je enkripcija korisničkih lozinki stoga se unutar baze podataka ne čuvaju direktno uneseni tekstualni zapisi lozinke. Prilikom registracije potvrđena lozinka šalje se na server u tekstualnom formatu pa servis obavlja enkripciju. Za enkripciju lozinki korištena je Googleova enkripcija biblioteka Tink. Implementacija enkripcije lozinki prikazana je ovdje:

```
fun initEncryption() {
        AeadConfig.register()
        val keysetFile = File(Secrets.KEYSET_FILENAME.value)
        if (keysetFile.exists()) {
            keysetHandle =
CleartextKeysetHandle.read(JsonKeysetReader.withFile(File(Sec
rets.KEYSET_FILENAME.value)))
        } else {
            keysetHandle =
KeysetHandle.generateNew(KeyTemplates.get("AES128_GCM"))
            CleartextKeysetHandle.write(keysetHandle,
JsonKeysetWriter.withFile(
File(Secrets.KEYSET_FILENAME.value)))
        }
    }
    fun encryptPassword(pass: String) : String {
        with(getHandle()) {
            val aead = getPrimitive(Aead::class.java)
            val encryptedText =
aead.encrypt(pass.toByteArray(Charsets.UTF_8), null)
            return
Base64.getEncoder().encodeToString(encryptedText)
        }}
    fun verifyPassword(pass: String, hash: String) : Boolean
{
```

```
            with(getHandle()) {
                val aead = getPrimitive(Aead::class.java)
                try {
                    val decrypted =
Base64.getDecoder().decode(hash)
                     val decryptedText = aead.decrypt(decrypted,null)
               return decryptedText.toString(Charsets.UTF_8) == pass
                }
                catch (ex: GeneralSecurityException){
                    println("Error decrypting: ${ex.message}")
                    return false
                }}}
```

Kod 4.16 Implementacija enkripcije korisničkih lozinki

Za enkripciju lozinki korištena je AEAD metoda. Sve spomenute metode nalaze se na singleton objektu `EncryptionManager`. Prilikom pokretanja servera poziva se metoda `initEncryption()` koja učitava enkripcijsku konfiguraciju. Unutar ove metode provjera se postoji li skrivena datoteka s generiranim enkripcijskim ključevima pa ili se učitava ili se generira nova. Dohvaćeni enkripcijski ključevi služit će i za enkripciju i dekripciju lozinki. Metoda za enkripciju lozinke prima lozinku u tekstualnom UTF-8 formatu i dohvaća njenu vrijednost u bitovima. Vrijednost u bitovima se zatim enkodira korištenjem ključeva, a generirani se niz dodatno enkodira u Base64 tekstualnu vrijednost koja se zatim sprema u bazu. Prilikom provjere lozinke zaprima se tekstualna vrijednost unesene lozinke iz klijentske aplikacije i enkriptirana vrijednost iz baze podataka vezana uz korisnički račun. Vrijednost iz baze najprije se Base64 dekodira pa prolazi dekripciju korištenjem enkripcijskih ključeva. Zadnji korak je provjera odgovara li dekriptirana vrijednost u UTF-8 formatu poslanoj lozinki.

5. Daljnji koraci i nadogradnja

Kao i sa svakim projektom u domeni programskog inženjerstva, nakon isporuke razvojni ciklus Budge sustava ne staje. Tijekom izrade projekata uvijek se ustanove određene komponente i mogućnosti koje bi se mogle implementirati bolje ili nadograditi na već postojeći sustav. Tržište mobilnih aplikacija iznimno je kompetitivno stoga je iznimno važno slušati reakcije korisnika vlastite i konkurentskih aplikacija kako bi se što bolje i brže odgovaralo na izazove tržišta. U ovome će se poglavlju predstaviti važnost praćenja analitike korištenja mobilnih aplikacija kao metode planiranja nadogradnji sustava. Na kraju, spomenut će se neki od potencijalnih smjerova u kojima bi se mogle kretati daljnje nadogradnje sustava Budge.

5.1. Analitika korištenja

Tržište mobilnih aplikacija iznimno je kompetitivan okvir pa se u svakome trenutku aplikacije bore za prevlast s tisućama drugih rješenja od razvojnih studija diljem svijeta. U svome istraživanju Tong Li, Hancheng Cao et al. obavili su detaljnu analizu navika korištenja aplikacija. Pratili su aktivnost 1465 korisnika u razdoblju od 2012. do 2017. te su bilježili broj aplikacija na mobitelu, kategorije aplikacija koje korisnici skidaju, broj aplikacija unutar istih kategorija i korištenje individualnih aplikacija. Za temu ovog rada najzanimljiviji su rezultati analize poredbe aplikacija unutar iste kategorije. Istraživanjem su utvrdili da svaka kategorija ima dva vremenska razdoblja: razdoblje rasta popularnosti nakon nastanka i plato najviše popularnosti. Broj aplikacija, koje se natječu unutar kategorije, smanjuje se postupno kako kategorija prelazi u drugu fazu. Ulaskom u drugu fazu, unutar kategorija se ističe nekoliko aplikacija koje dominiraju preko 80% prometa u kategoriji. U ovoj drugoj fazi najznačajniji utjecaj na zadržavanje i preotimanje korisnika imaju inovativne funkcionalnosti. Ako korisnici ne mogu pronaći sve funkcionalnosti koje ih zanimaju, koristit će više aplikacija u istoj kategoriji, ali sveobuhvatna rješenja imaju veće šanse zadržati korisnike [5]. Kao što je vidljivo, iznimno je važno prepoznati želje svojih korisnika i pratiti što konkurenti rade pravilno. Jason Wong u blogu za svjetski poznatu konzultantsku tvrtku Gartner navodi da je vođenje analitike korištenja mobilnih

aplikacija jedna od osnovnih komponenti životnog ciklusa razvoja aplikacije [6]. Analitika korištenja omogućava dublje razumijevanje ponašanja korisnika i način na koji koriste aplikacije, što omogućava analizu efektivnosti implementiranog rješenja koja u konačnici rezultira operativnim planovima za poboljšanje proizvoda i korisničkog iskustva. Vrlo je bitno da se prikupljeni podaci koriste za kreiranje strategije razvoja povezane s poslovnim ciljevima. Pravilnim korištenjem saznanja moguće je značajno poboljšati stopu zadržavanja korisnika i generalno korisničko iskustvo.

Za izvedbu prikupljanja analitike u projektu Budge, korišten je sustav Google Firebase Analytics. Na Google Firebase portalu kreiran je projekt i pokrenut modul za analitiku. Firebase sustav generira sve potrebne konfiguracijske datoteke koje se jednostavno uključuju u već postojeću aplikaciju. Korištenjem dohvaćenih klasa omogućena je direktna komunikacija aplikacije s Firebase Analytics web-servisom. Analytics modul omogućava bilježenje korisničkih interakcija kao kombinacije analitičkih ključeva pod kojim se čuvaju događaji iz aplikacije i određene tekstualne ili brojevne vrijednosti. Vezanjem analitičkih događaja na gumbe za CRUD operacije i interakcije moguće je stvoriti sliku svih mogućnosti aplikacije koje korisnici najviše koriste. Događaji, koji imaju mali broj ponavljanja, signaliziraju da korisnici nisu zadovoljni implementacijom i da se nakon inicijalnog korištenja više ne vraćaju na određene ekrane. Sve zabilježene evente iz aplikacije moguće je detaljno proučavati na Google Firebase portalu. Firebase Analytics portal omogućava grafički prikaz i analizu događaja uz dodatne mogućnosti filtriranja događaja prema zadanim parametrima (npr. je li korisnik novi ili postojeći).

Nakon puštanja u produkciju, plan bi bio nakon nekog vremenskog razdoblja (nekoliko mjeseci), kada bi se sakupio relevantan broj podataka, početi obavljati analizu korisničke interakcije. Detaljnijom analizom moguće je primijetiti koje su mogućnosti aplikacije popularne, a koje zanemarene od strane korisnika, što omogućava fokus na razvoj i poboljšanje onih mogućnosti koje korisnici stvarno žele.

5.2. Potencijalne nadogradnje

Tijekom izrade projekta ističu se detalji implementacije planiranih mogućnosti u praktičan sustav. Samim time sakupljaju se nova saznanja i na kraju svakog projekta generiraju se ideje što je moguće izvesti bolje, brže, logičnije ili elegantnije. U slučaju projekta Budge dostupan je još jedan širok izvor saznanja: analitika korisničke interakcije s aplikacijom.

Izradom sustava pojavila su se određena područja koja bi se mogla dalje i detaljnije razviti i nadopuniti novim mogućnostima. U ovome potpoglavlju će se spomenuti neka od njih. Također, promotrit će se moguće monetizacijske politike za daljnji tijek projekta.

Konkurentske aplikacije Revolut i Wallet nude opcije kategoriziranja svoje potrošnje kroz velik broj predefiniranih kategorija potrošnje novca. Ovakav način implementacije nudi korisnicima elegantan način raspodjele mjesečnih primanja te im omogućava užu kontrolu svoje mjesečne potrošnje. Zbog načina implementacije tablica u bazi podataka i načina kalkuliranja projekcija, ovu mogućnost nije bilo jednostavno bez većih preinaka isporučiti u inicijalnoj verziji. Stoga, ona stoji kao jedna od prvih nadogradnji.

Mogućnosti baratanja investicijama u aplikaciji trenutno su limitirane na pregled trenutnog srednjeg stanja vrijednosti na tržištu. Konkurirati Revolutu kao aplikacija za trgovanje nije nimalo jednostavno niti su šanse za dominaciju izgledne. Stoga, dobar smjer razvoja bio bi fokus na detaljnije istraživanje kretanja investicijskih tržišta. Pregled cijena na različitim mjenjačnicama, detaljan pregled indeksa i različitih dnevnih cijena dionica, implementacija projekcija kretanja cijena investicija na tržištu samo su neke od mogućnosti daljnjeg razvoja sustava.

Detaljna analiza podataka prikupljenih od strane korisnika omogućit će izradu pregleda korištenja aplikacije. Pregledom sesije prosječnog korisnika moći će se ustanoviti koje se mogućnosti u aplikaciji najviše koriste, koje su zanemarene, koje se brzo napuštaju te koje mogućnosti, prema analitici zatvaranja i nekorištenja, frustriraju korisnike. Prateći ove parametre i komentare korisnika na trgovinama za mobilne aplikacije, izradit će se detaljan plan nadogradnje sustava kako bi se razvijali točno oni dijelovi sustava koje korisnici najviše žele.

Originalna monetizacijska politika sustava Budge je da bude besplatan. Osnovna je ideja ovog projekta pomoći mladoj populaciji ovladati osobnim financijama i kod njih potaknuti zanimanje za razvoj financijske pismenosti. Nažalost, tržište ne mari za dobre želje razvojnih studija pa je potrebno pokrivati operacijske troškove sustava. Ako bi bilo potrebno početi monetizirati Budge, postoji nekoliko monetizacijskih smjerova. Moguće je osnovne funkcionalnosti ostaviti besplatne, dok bi se sav naknadno dodani sadržaj zatvarao iza premium inačice koja bi se naplaćivala. U slučaju da se investicijske mogućnosti pokažu popularnima, bilo bi moguće monetizirati detaljnu analizu tržišta. Ne isključuju se

ni opcije suradnje s konkurentima poput Revoluta (npr. uvezivanje računa s Revoluta) kako bi se ostvarila zajednička dobit.

Zaključak

Modernizacija i digitalizacija svijeta omogućile su velik broj opcija čuvanja i investiranja kapitala. Mnoge mlade osobe kroz svoje obiteljsko i formalno obrazovanje ne stječu zadovoljavajuću razinu financijske pismenosti i pate od financijskih problema uzrokovanih neznanjem i dezorganizacijom. Rješenja ovog problema pronalaze se u digitalnim alatima koji pojednostavljuju baratanje osobnim financijama i stvaraju kod korisnika disciplinu i osjećaj za raspolaganje novcem i kapitalom.

Trenutno tržište aplikacija za osobne financije još je otvoreno i zasad nije ponuđeno sveobuhvatno i dominantno rješenje. Ideja ovog rada bila je predložiti jedno takvo potencijalno rješenje koje kroz mnoštvo dodatnih opcija nastoji poboljšati i obogatiti korisničko iskustvo te olakšati i ubrzati proces vođenja osobnih financija.

Kroz izradu projekta proučeni su moderni principi i arhitekture izrade web-servisa i mobilnih aplikacija. Za izradu programskog rješenja, korištena je troslojna sistemska arhitektura sazdana od podatkovnog sloja, sloja poslovne logike i klijentskog sloja. Istražene su opcije zaštite i pohrane korisničkih podataka. Proučen je značaj korištenja analitike korisničkog iskustva unutar aplikacija te je jedan takav sustav i implementiran u rješenje korištenjem tehnologije Google Firebase Analytics. Prilikom izrade aplikacije korišteni su najmoderniji pristupi za izradu mobilnih aplikacija – MVVM i nova tehnologija za deklarativno stvaranje korisničkog sučelja – Jetpack Compose. Izrada web-servisa korištenjem tehnologije Ktor stvorila je priliku za implementaciju modernih pristupa i metodologija izrade servisa.

Rezultat je ovog rada funkcionalan sustav koji bi zaista mogao pomoći ljudima pri planiranju i vođenju osobnih financija. Određene prepreke uvođenja aplikacije u produkcijsko okruženje, dakako, postoje. Integracija sa stvarnim bankarskim sustavima zahtijevala bi mnogo truda i podizanje standarda enkripcije i sigurnosti sustava. Tijekom izvođenja rada detektirane su određene značajke koje bi se mogle poboljšati u dogledno vrijeme te postoji opcija da se sustav Budge nastavi razvijati kao javni ili hobi projekt. Cijeli proces izrade ovoga programskog rješenja poslužio je kao izvrsna prilika za upoznavanje s novim tehnologijama i alatima te kao prilika za proučavanje i usavršavanje pristupa programskog inženjerstva na većem i realnom projektu.

Popis kratica

AEAD	*Authenticated Encryption with Associated Data*
API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
LDAP	*Lightweight Directory Access Protocol*
MVVM	*Model-View-ViewModel*
ORM	*Object Relational Mapping*
OS	*Operating system*
REST	*Represemtational state transfer*
SQL	*Structured query language*
UI	*User interface*
URL	*Universal Resource Link*
XML	*Extensible markup language*

Popis slika

Popis kôdova

Literatura

[1] Malton, C., Clarkson, T. (2017.) "Young adults and money management: behaviours, attitudes and useful rules of thumb" BritainThinks

[2] Sinha, G., Tan, K., Zhan, M. (2018.) " Patterns of financial attributes and behaviors of emerging adults in the United States" Children and Youth Services Review, 2018; 93

[3] Android Developers: Guide to App architecture (2022.) Dostupno na: https://developer.android.com/topic/architecture [2. veljače 2023.]

[4] Scholarly Community Encyclopedia: Epoch (2022.) Dostupno na: https://encyclopedia.pub/entry/31703 [5. veljače 2023.]

[5] Li, T., Zhang, M., Cao, H., Li, Y., Tarkoma, S. (2020.) "What Apps Did You Use?: Understanding the Long-term Evolution of Mobile App Usage" The Web Conference 2020 (WWW'20) 20-24. travnja 2020. Taipei

[6] Wong, J. (2014.) „The Importance of Mobile App Analytics" Gartner Dostupno na:

https://blogs.gartner.com/jason-wong/the-importance-of-mobile-app-analytics [9. veljače

2023.]

[7] Jemerov, D., Isakova, S. (2017.) "Kotlin in action" Manning

[8] Kleppmann, M. (2017.) " Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems " O'Reilly Media

[9] Künneth, T. (2022.) " Android UI Development with Jetpack Compose " Packt Publishing

[10] Stojanović, S. (2023.) "Fundamentals of Compose layouts and modifiers" Android Developers Dostupno na: https://medium.com/androiddevelopers/fundamentals-of-compose-layouts-and-modifiers-64d794664b66 [9. veljače 2023.]

[11] Stamato, A. (2022.) "Effective state management for TextField in Compose" Android Developers Dostupno na: https://medium.com/androiddevelopers/effective-state-management-for-textfield-in-compose-d6e5b070fbe5 [9. veljače 2023.]