Mobilna aplikacija za aktivno uključivanje djece u svakodnevne aktivnost

Gotal, Davor

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:637162

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-16**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

MOBILNA APLIKACIJA ZA AKTIVNO UKLJUČIVANJE DJECE U SVAKODNEVNE AKTIVNOSTI

Završni rad

Davor Gotal

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Davor Gotal
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4488, 27.07.2020.
OIB Pristupnika:	27887437743
Mentor:	izv. prof. dr. sc. Josip Balen
Sumentor:	Matej Arlović, mag. ing. comp.
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna aplikacija za aktivno uključivanje djece u svakodnevne aktivnost
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U Završnom radu potrebno je izraditi mobilnu aplikaciju za aktivno uključivanje djece u obavljanje u svakodnevnih aktivnost poput kućanskih poslova putem osmišljenih kreativnih i poticajnih zadataka za djecu, uključivanjem gemifikacije i sustava nagrađivanja. U aplikaciju je potrebno ugraditi profiliranje korisnika prema ulozi, CRUD operacija nad svim zadacima, pretraživanje i filtriranje podataka, sustav bodovanja i rangiranje pohranjivanje
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2023.
Datum potvrde ocjene od strane Odbora:	08.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.09.2023.

Ime i prezime studenta:	Davor Gotal
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4488, 27.07.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za aktivno uključivanje djece u svakodnevne aktivnost**

izrađen pod vodstvom mentora izv. prof. dr. sc. Josip Balen

i sumentora Matej Arlović, mag. ing. comp.

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. KORIŠTENE TEHNOLOGIJE 3

2.1. Android Studio razvojno okruženje 3

2.2. Programski jezik Kotlin 4

2.3. Jetpack Compose 4

2.4. Firebase 6

3. RAZVOJ APLIKACIJE 8

3.1. Grafičko sučelje 8

3.2. Izrada baze podataka 9

3.3. Upravljanje korisnicima 10

3.3.1. Prijava korisnika u aplikaciju 10

3.3.2. Registracija korisnika 11

3.3.3. Ponovno postavljanje korisničke zaporke 12

3.3.4. Odjava korisnika 12

3.4. Početni korisnički zasloni 12

3.4.1. Prikaz zaslona korisnika uloge roditelja 13

3.4.2. Prikaz zaslona korisnika uloge djeteta 14

3.5. Upravljanje zadacima 15

3.5.1. Dodavanje zadataka 16

3.5.2. Brisanje zadataka 16

3.5.3. Potvrda zadataka 17

3.5.4. Slanje zadatka na ponovni pregled 18

3.5.5. Uređivanje zadataka 20

3.5.6. Pretraživanje zadataka 21

3.5.7. Automatsko ažuriranje zadataka 21

3.6. Upravljanje željama 22

3.6.1. Dodavanje želja 22

3.6.2. Brisanje želja 23

3.6.3. Uređivanje želja 23

4. ZAKLJUČAK 24

LITERATURA **25**

SAŽETAK **26**

ABSTRACT **27**

ŽIVOTOPIS **28**

PRILOZI **29**

4.1. Izvorni kod aplikacije **29**

1. UVOD

S digitalizacijom svijeta djeca sve manje sudjeluju u fizičkim aktivnostima svakodnevnog života. Ova promjena ima potencijalno negativne posljedice, budući da se djeca koja odrastaju bez iskustva s ovom vrstom aktivnosti često kasnije u životu suočavaju s nedostatkom praktičnih vještina i neovisnosti. Osim toga, problem nedostatka praktičnih vještina mogao bi utjecati na buduće generacije, jer neće biti roditelja koji bi to znanje mogli prenijeti na mlađe generacije. U modernim obrazovnim sustavima često nedostaje naglasak na praktičnim vještinama koje su ključne za uspješan i neovisan život. Većina škola prvenstveno se fokusira na prenošenje teoretskog znanja i postizanje akademske izvrsnosti, dok se praktična primjena tog znanja često zanemaruje. Nedostatak prakse i praktičnog iskustva u obavljanju svakodnevnih zadataka može učiniti da se mladi ljudi osjećaju nepripremljeno ili nesigurno u situacijama stvarnog života.

Kroz aktivno sudjelovanje u fizičkim poslovima, djeca razvijaju vještine samostalnosti, odgovornosti, timskog rada i praktičnih vještina. Navedeni poslovi pružaju priliku za razumijevanje vrijednosti rada, truda i postignuća što zbog današnjih okolnosti mladi imaju sve manju priliku iskusiti. Kao rezultat toga, sve češće susrećemo primjere mladih ljudi koji nisu osposobljeni za obavljanje jednostavnih zadataka ili nemaju potrebne vještine za svakodnevne aktivnosti. Zbog nespremnosti na samostalan život, mladi sve duže ostaju živjeti kod svojih roditelja i kada su dovoljno odrasli za samostalan život. Pojava dugotrajnog boravka mladih s roditeljima može imati negativne posljedice ne samo za mladu osobu nego i za društvo u cjelini.

Jedan od načina rješavanja nedostatka praktičnih vještina kod mladih je korištenje moderne tehnologije i digitalnih alata. Upravo zbog toga je razvijena android aplikacija za uključivanje mladih u svakodnevne aktivnosti koja će mladima dati priliku za razvoj samostalnosti, odgovornosti, timskog rada i drugih praktičnih vještina kroz sustav nagrađivanja. Korištenjem aplikacije mladi će imati priliku povećati svoju svijest o vrijednosti truda i rada. Kako bi se privuklo mlade na korištenje aplikacije za uključivanje mladih u svakodnevne aktivnosti, aplikacija koristi posebno odabranu paletu boja. Boje su bitan faktor u privlačenju i zadržavanju dječje pažnje te iz tog razloga su korištene boje kao što su svijetlo plava, narančasta i druge vesele nijanse koje pridonose zabavnom i interaktivnom korisničkom iskustvu aplikacije i potiče mlade da aktivno sudjeluju u zadacima i aktivnostima.

1.1. Zadatak završnog rada

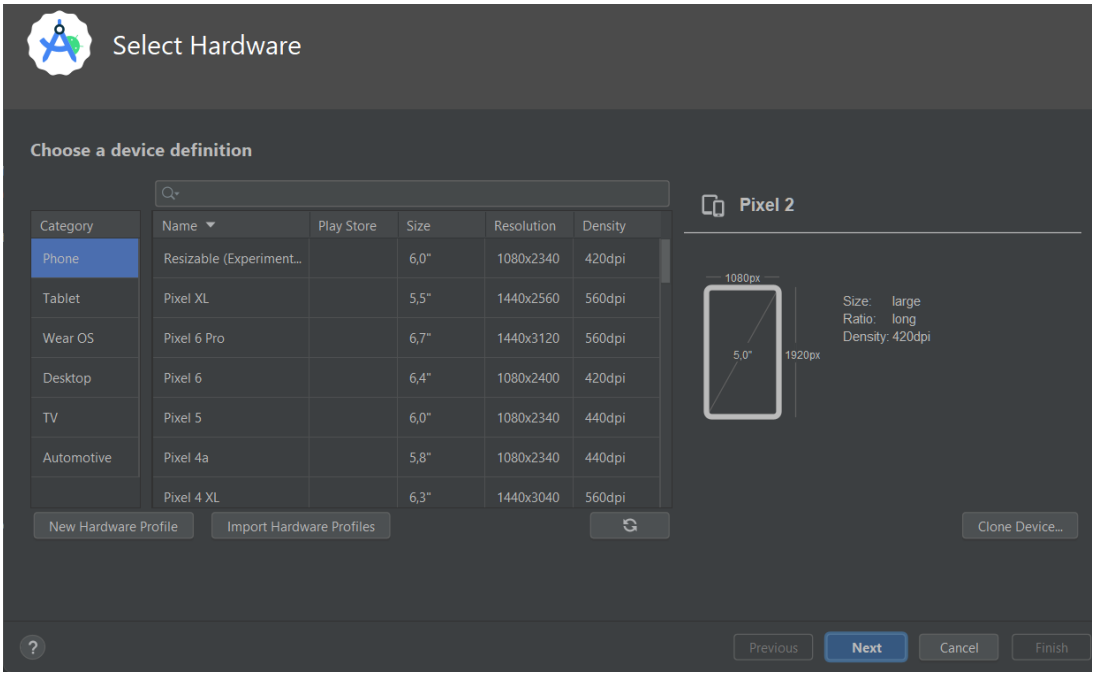
Zadatak završnog rada je napraviti android aplikaciju za aktivno uključivanje mladih u svakodnevne aktivnosti koja bi poticala mlade da stječu praktične vještine uz odgovarajući sustav nagrađivanja. Aplikacija će imati dvije razine pristupa za roditelje i djecu. Roditelj će dodjeljivati djeci poslove i za svaki posao će unijeti određenu količinu digitalnog novca koju smatra adekvatnom za taj posao. Djeca će moći vidjeti ponuđene poslove koje nakon što obave mogu poslati na provjeru roditeljima. Također, djeca će moći davati prijedloge za listu želja koju će roditelji odobriti i postaviti količinu digitalnog novca kojeg smatraju adekvatnim.

2. KORIŠTENE TEHNOLOGIJE

Android aplikacija će biti izrađena u Android Studio razvojnom okruženju. *Android Studio* je integrirano razvojno okruženje koje je namijenjeno za razvoj aplikacija za Android platformu. Uz Android Studio moguće je koristiti *Java* ili *Kotlin* programski jezik za pisanje programskog koda. Svrha ove aplikacije je potaknuti mlađu populaciju na aktivno sudjelovanje u svakodnevnim aktivnostima kako bi stekli potrebne vještine za samostalan život u budućnosti putem dodjeljivanja zadataka kroz aplikaciju koji ako su uspješno obavljeni će biti adekvatno nagrađeni sustavom nagrađivanja kojeg roditelj osmišljava.

2.1. Android Studio razvojno okruženje

Android Studio je integrirano programsko okruženje za razvoj Android aplikacija razvijeno od strane Google kompanije. Android Studio podržava različite programske jezike za razvoj Android aplikacija. Najčešće korišteni programski jezici su *Java*, *Kotlin*, C++, C#, *JavaScript* i *Python*[1]. Osim pisanja koda, Android Studio razvojno okruženje nudi skup alata za vizualni dizajn korisničkog sučelja kao i ugrađeni emulator koji programerima omogućuje testiranje svojih aplikacija na različitim virtualnim uređajima te nije potrebno fizički posjedovati sve uređaje, već je omogućeno testiranje na novim i starim uređajima aplikacije što se može vidjeti na slici 2.1. Emulator omogućuje simulaciju različitih karakteristika uređaja kao što su rezolucija, veličina zaslona, verzija operativnog sustava Android[2].



Slika 2.1. Kreiranje virtualnog uređaja

2.2. Programski jezik Kotlin

Programski jezik *Kotlin* je moderan programski jezik koji programerima nudi brojne prednosti za povećanje njihove produktivnosti i smanjenja vremena utrošenog na pisanje i održavanje istog koda za različite platforme, zadržavajući pritom fleksibilnost i prednosti izvornog programiranja.

Dizajnirana za pojednostavljenje razvoja projekata na više platformi *Kotlin Multiplatform* tehnologija omogućuje dijeljenje koda između *iOS* (engl. *iPhone operating system*) i Android sustava za stvaranje mobilnih aplikacija koje dijele zajedničku implementaciju za različite funkcije kao što su provjera valjanosti podataka, pohrana podataka, umrežavanje, analiza, računanje i druge logike aplikacije.[3]

Uz *Compose Multiplatform* platformu, deklarativni okvir korisničkog sučelja, moguće je dijeliti korisnička sučelja između *Android* i *iOS* operacijskih sustava. To omogućava izgradnju aplikacija za više platformi koje dijele programski kod i korisničko sučelje između mobilnih platformi. Ova platforma omogućuje programerima da objedine korisničko iskustvo na svim uređajima, osiguravajući dosljedan izgled i dojam aplikacije, jednostavno testiranje, održavanje i iterativno razvijanje aplikacije na više platformi.[4]

2.3. Jetpack Compose

Jetpack Compose je moderni alat za izradu korisničkog sučelja u Android aplikacijama. Koristi se u raznim poznatim aplikacijama poput *Pinterest*, *SoundCloud* i *Lyfata* [5]. Tradicionalni pristup programiranju sučelja putem XML (engl. *Extensible Markup Language*) datoteka je zamijenjen *Kotlin* DSL (engl. *Domain-Specific-Language*) pristupom koji programski kod čini čitljivijim, održivim i smanjuje mogućnost pogrešaka. Promjene u programskom kodu se trenutno prikazuju u emuliranom ili stvarnom uređaju što omogućuje programerima da trenutno vide rezultate izmjena u programskom kodu te time ubrzaju razvojni proces i eksperimentiranje s različitim idejama izgleda sučelja [6].

Composable funkcije su ključne značajke za izradu korisničkog sučelja. Omogućavaju deklarativno definiranje dijelova korisničkog sučelja u obliku funkcija, što olakšava ponovnu upotrebu i održavanje koda. Primjer ponovno upotrebljive funkcije korištene u ovom radu pod

nazivom *InputField* čiji kod je prikazan u programskom kodu 2.1., a korištena je za unos podataka prilikom prijave i registracije korisnika.

```
@Composable
fun InputField(
    value: String,
    label: String,
    placeholder: String,
    visualTransformation: VisualTransformation =
VisualTransformation.None,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    leadingIcon: @Composable (() -> Unit)? = null,
    trailingIcon: @Composable (() -> Unit)? = null,
    onValueChange: (String) -> Unit
) {
    OutlinedTextField(
        value = value,
        onValueChange = onValueChange,
        label = {
            Text(text = label)
        },
        placeholder = {

        },
        visualTransformation = visualTransformation,
        keyboardOptions = keyboardOptions,
        leadingIcon = leadingIcon,
        trailingIcon = trailingIcon
    )

}
```

Programski kod 2.1. InputField composable funckija

Korištenjem *Jetpack Compose* omogućen je komponibilni pregled koji omogućava programerima vizualiziranje i testiranje *Compose* komponenata prije nego što se integriraju u aplikaciju što ubrzava razvoj aplikacije, budući da nije potrebno pokretati emulator ili instalirati aplikaciju na stvarnom uređaju, što često zahtijeva više vremena[7]. Primjer takvog pristupa je funkcija *AvatarItemPreview()* koja je prikazana u programskom kodu 2.2. Funkcija *AvatarItemPreview()* omogućava vizualizaciju okvira profilne slike kao i samu sliku.

```
@Preview
@Composable
fun AvatarItemPreview() {
    AvatarItem(
        isSelected = true,
        onAvatarClick = {},
        avatarUrl =
"https://firebasestorage.googleapis.com/v0/b/helpandearn-
c6485.appspot.com/o/images%2Favatar1.png?alt=media&token=526a63cc-0187-
4b23-802e-30e37e0d9b32"
    )
}
```

Programski kod 2.2. Komponibilni prikaz profilne slike

2.4. Firebase

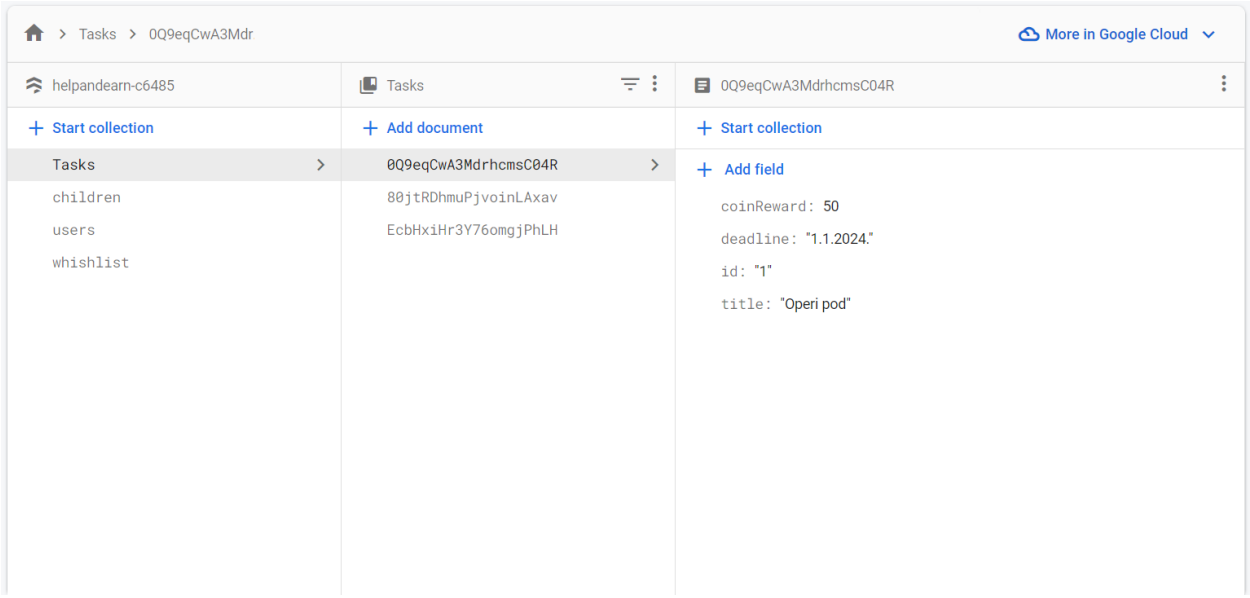
Firebase je platforma razvijena od strane *Google* koja je namijenjena upravljanju i razvoju web i mobilnih aplikacija. Sastoji se od raznih usluga i alata koji olakšavaju razvoj, analizu, nadzor i testiranje aplikacija[8]. *Firebase Realtime Database* je baza podataka koja omogućava istovremeno ažuriranje podataka između *Firebase* poslužitelja i klijentskih aplikacija u stvarnom vremenu. Ova tehnologija je posebno korisna za mobilne aplikacije koje zahtijevaju trenutna i neprekidna ažuriranja. Također, *Firebase* pruža i uslugu *Firebase Authentication* koja pruža autentifikaciju korisnika koja omogućava registraciju i prijavu korisnika putem različitih metoda, kao što su e-pošta i lozinka, Google prijava ili prijava putem društvenih mreža. Postoje još razne usluge koje pruža *Firebase* platforma poput *Firebase Cloud Messaging*, alate za analitiku i praćenja performansi aplikacija.

Firebase omogućuje definiranje pravila baze podataka kao što su dozvole pristupa i pisanja za dokumente. Pravila ograničavaju pristup i izmjenu podataka unutar *Firestore* baze podataka tako da osigurava pristup samo korisnicima koji su prijavljeni u aplikaciju. U prikazanim pravilima u programskom kodu 2.3. je dozvoljeno čitanje dokumenata unutar kolekcije svim korisnicima dok za izmjenu dokumenata unutar kolekcije je moguće jedino ako je korisnik potvrđen.

```
rules_version = '2';

service cloud.firestore {
    match /databases/{database}/documents {
        match /Tasks/{taskId} {
            allow read: if request.auth != null;
            allow write: if request.auth != null;
        }
        match /children/{childId} {
            allow read: if request.auth != null;
            allow write: if request.auth != null;
        }
        match /users/{documentId} {
            allow read: if request.auth != null;
            allow write: if request.auth != null;
        }
          match /wishlist/{documentId} {
            allow read: if request.auth != null;
            allow write: if request.auth != null;
          }
              match /{document=**} {
                  allow read, write: if false;
              }
          }
      }
```

Programski kod 2.3. Definiranje pravila pristupa

Firebase Firestore baza podataka koristi dokumente i kolekcije kako bi organizirala podatke. Dokumenti su osnovna jedinica podataka koje se pohranjuju u kolekcije. Svaki dokument ima jedinstveni identifikator koji se naziva i putanja dokumenta. Na slici 2.2. prikazana je Firestore baza podataka koja sadrži nekoliko kolekcija tasks, children, users, whishlist. Navedene kolekcije se mogu dohvaćati, mijenjati ili brisati.



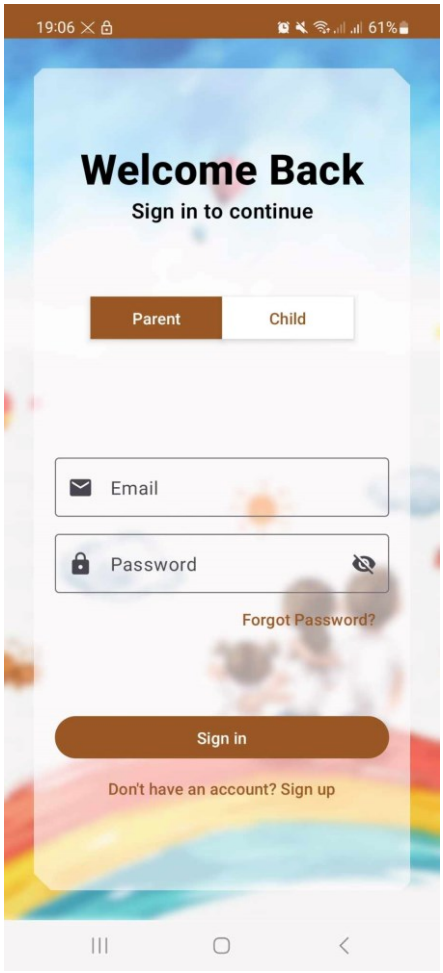
Slika 2.2. Prikaz baze podataka

3. RAZVOJ APLIKACIJE

Razvoj aplikacije složen je proces koji zahtijeva planiranje, implementaciju i testiranje kako bi se postigla željena funkcionalnost i dizajn. Ovo poglavlje detaljno opisuje korake i postupke koji su uključeni u razvoj Android aplikacije. Istaknute su najvažnije faze razvoja, korišteni alati i tehnologije. Naglasak je na najvažnijim dijelovima koda koji su odgovorni za ispunjavanje same svrhe aplikacije.

3.1. Grafičko sučelje

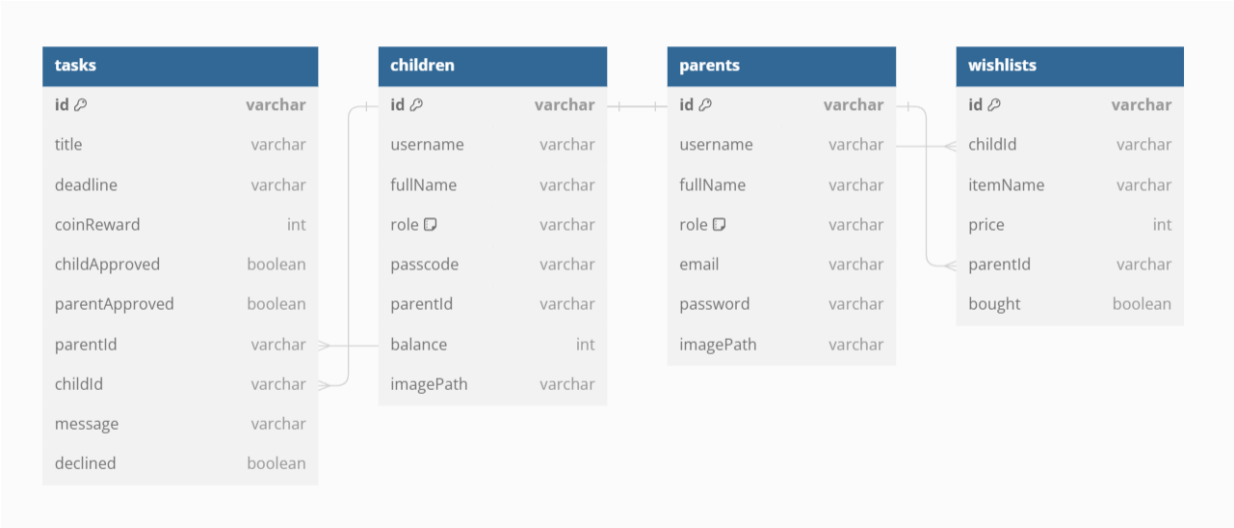
Grafičko sučelje predstavlja ključni dio mobilne platforme koji omogućava korisnicima interakciju s aplikacijama na njihovim mobilnim uređajima. Sastoji se od intuitivnih elemenata poput ikona, gumba, slika, teksta i drugih elemenata korisničkog sučelja. Početni zaslon aplikacije koji prikazuje prijavu korisnika u aplikaciju je prikazan na slici 3.1.



Slika 3.1. Izgled početnog zaslona

3.2. Izrada baze podataka

Baza podataka prikazana na slici 3.2. prikazuje relacije između četiri glavna entiteta: *Task*, *Child*, *Parent* i *Wishlist*. Entiteti su povezani relacijama koje omogućuju organizaciju i praćenje veza između roditelja, djece, njihovih zadataka i želja.



Slika 3.2. Baza podataka

Relacija između entiteta zadatak i dijete koji predstavljaju dijete i njemu dodijeljen zadatak omogućuje povezivanje pojedinih zadataka s određenim djetetom putem atributa *childId*. Zadatak može biti dodijeljen samo jednom djetetu, dok jedno dijete može imati više zadataka. Ova veza ima ključnu ulogu u praćenju odgovornosti djeteta za izvršavanje određenog zadatka. Tako, kroz bazu podataka, moguće je lako pratiti koja su djeca zadužena za koje zadatke i koji zadaci su im trenutno na raspolaganju. Također, relacija omogućuje slanje zadatka na pregled roditelju. Kada dijete dovrši zadatak i pošalje ga na pregled, roditelj može provjeriti njegovo izvršenje i donijeti odluku o odobravanju ili odbijanju zadatka. Ova veza između zadatka i dijete entiteta ključna je za uspješno upravljanje zadacima unutar sustava i olakšava komunikaciju između roditelja i djece u pogledu raspodjele i praćenja zadatka.

Entitet zadatak i roditelj su povezani relacijom koja omogućuje povezivanje zadatka s odgovarajućim roditeljem. Prema definiranoj relaciji, svaki zadatak može biti povezan samo s jednim roditeljem, dok jedan roditelj može imati više zadataka koje je dodijelio određenom djetetu. Ova veza pruža roditelju mogućnost dodjeljivanja zadataka svojoj djeci i praćenje njihovog napretka. Relacija omogućuje roditelju da odobri ili odbije izvršenje zadatka od strane djeteta. To olakšava komunikaciju između roditelja i djeteta o zadacima, a roditeljima omogućuje potpunu kontrolu i pregled zadataka dodijeljenih njihovom djetetu.

Relacija između entiteta dijete i roditelj uspostavlja vezu između djeteta i odgovarajućeg roditelja. Prema definiranoj relaciji, svako dijete ima samo jednog roditelja, dok jedan roditelj može imati više djece. Ova veza omogućuje praćenje obiteljske pripadnosti djeteta i organizaciju podataka o roditeljima i njihovoj djeci. Kroz ovu vezu, roditelj može jednostavno pristupiti informacijama o svom djetetu i upravljati zadacima koje je dodijelio toj osobi. Također, omogućuje se jednostavna navigacija kroz podatke o obitelji i olakšava se izvršavanje operacija koje uključuju roditelje i njihovu djecu unutar sustava upravljanja zadacima.

Entiteti dijete, roditelj i želja su povezani putem *parentId* i *childId* atributa. Svako dijete može imati više različitih želja, dok s druge strane svaka želja je povezana s jednim roditeljem putem *parentID* atributa i jednim djetetom putem *childId* atributa. Kroz ovu relaciju roditelj može dodijeliti želju djetetu stvarajući vezu između djeteta, roditelja i želje što omogućava korisniku roditelj praćenje napretka djeteta u ostvarivanju želja.

3.3. Upravljanje korisnicima

Upravljanje korisnicima unutar aplikacije ključni je dio funkcionalnosti koji korisnicima omogućuje pristup personaliziranom sadržaju i funkcionalnosti. U implementaciji aplikacije korisnici se mogu prijaviti u aplikaciju sa stalnim ulogama "roditelj" i "dijete" te se mogu registrirati kao novi korisnici. Također, postoji opcija brisanja korisnika iz *Firebase* baze podataka. Funkcionalnost prijave korisnika u aplikaciji temelji se na platformi *Firebase* koja pruža sigurnu autentifikaciju putem e-pošte i lozinke. Ovisno o ulozi korisnika, oni imaju različita prava pristupa funkcijama i sadržaju unutar aplikacije.

3.3.1. Prijava korisnika u aplikaciju

Prijava korisnika u aplikaciju je prva interakcija korisnika s aplikacijom te predstavlja ključni korak u korisničkom iskustvu. U programskom kodu 3.1. prikazana je implementacija funkcije prijave korisnika u aplikaciju. Funkcija koristi postupak prijave korisnika putem *Firebase Authentication* sustava. U slučaju uspješne prijave korisnik će prema svojoj ulozi koja može biti roditelj i dijete biti proslijeđena na odgovarajući zaslon aplikacije. Početni korisnički zasloni prema ulozi korisnika će biti obrađeni u nastavku rada.

```
fun login(email: String, password: String, onLoginSuccess: (String) ->
Unit) {
    val auth = FirebaseAuth.getInstance()
    signInWithEmail(auth, email, password, onLoginSuccess)
}
```

Programski kod 3.1. Funkcija za prijavu korisnika

3.3.2. Registracija korisnika

Registracija roditelja u aplikaciji je predstavljena funkcijom *registerUser()* koja prima osnovne podatke od korisnika kao što su korisničko ime, puno ime, uloga, e-mail adresa i lozinka. U funkciji prikazanoj u programskom kodu 3.2. se stvara nova instanca objekta roditelj s prikupljenim podacima korisnika i jedinstvenim identifikatorom. Stvoreni objekt sadrži sve važne informacije o korisniku. Nakon što se roditelj uspješno registrira pojavit će se poruka o uspješnom kreiranju računa. U suprotnom će se prikazati poruka o neuspješnoj registraciji.

```
fun registerUser(parent: Parent) {
    viewModelScope.launch {
        try {
            val userId = auth.createUserWithEmailAndPassword(parent.email,
parent.password)
                .await()
                .user
                ?.uid
            if (userId != null) {
                parent.id = userId
                usersCollectionRef.document(userId).set(parent).await()
            }
            _registrationSuccess.value = true
            println("User registered successfully with ID: ${parent.id}")
        } catch (e: Exception) {
            _registrationSuccess.value = false
            println("Error registering user: ${e.message}")
        }
    }
}
```

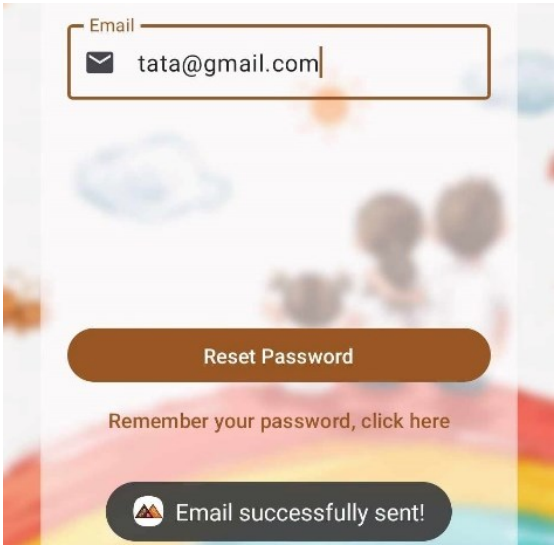
Programski kod 3.2. Funkcija za registraciju korisnika

Navedena funkcija odgovorna za registraciju korisnika u aplikaciji se izvodi asinkrono koristeći paralelni proces (engl. coroutine) koja se pokreće pozivom funkcije *launch*. *ViewModelScope* je dio arhitekture koji služi za upravljanje životnim ciklusom *ViewModela* na način da ukoliko se napusti zaslon registracije pokrenuti paralelni proces se automatski otkazuje zbog sprječavanja curenja resursa.

Registracija korisnika čije uloga je dijete je definirana na način da roditelj dodaje djecu u aplikaciju unosom imena, korisniče oznake i brojčane šifre. Prilikom poziva funkcije za dodavanje djece u bazu podataka se pohranjuju unesi podaci zajedno s jedinstvenim identifikatorom roditelja te se kreira jedinstveni identifikator djeteta. Ukoliko se pokuša kreirati korisnički račun djeteta s već unesenom korisničkom oznakom ili ukoliko sva polja za unos nisu popunjena pojavit će se poruka upozorenja te se neće pozvati funkcija za kreiranje dok roditelj ne unese ispravne podatke.

3.3.3. Ponovno postavljanje korisničke zaporke

Ponovno postavljanje korisničke zaporke je implementirano putem *Firebase* platforme. Korisnik na početnom zaslonu aplikacije odabire opciju ponovnog postavljanje zaporke koji ga preusmjerava na novi zaslon u kojem unosi svoju elektroničku poštu s kojom je kreirao korisnički račun u aplikaciji. Nakon potvrde prima obavijest *Toast* poruke o uspješnom zahtjevu za ponovno postavljanje zaporke.



Slika 3.3. Zaslon ponovnog postavljanja zaporke

3.3.4. Odjava korisnika

Korisnik putem navigacijske trake se može preusmjeriti na zaslon profila gdje ima opciju odjave iz aplikacije koja poziva funkciju *clearUserData*() prikazanu na programskom kodu 3.3. Unutar ove funkcija, prvo se poziva metodu *clearUserParentData*() iz *preferencesManager* objekta koja će obrisati sve podatke o korisniku, a zatim se poziva *signOut*() metoda iz *firebaseAuth* objekta koja će odjaviti korisnika iz njegovog računa. Korištenjem *clearUserData*() funkcije potpuno se uklanjaju korisnički podaci i omogućuje se sigurna odjava iz aplikacije

```
fun clearUserData() {
    preferencesManager.clearUserParentData()
    firebaseAuth.signOut()
}
```

Programski kod 3.3. Odjava korisnika

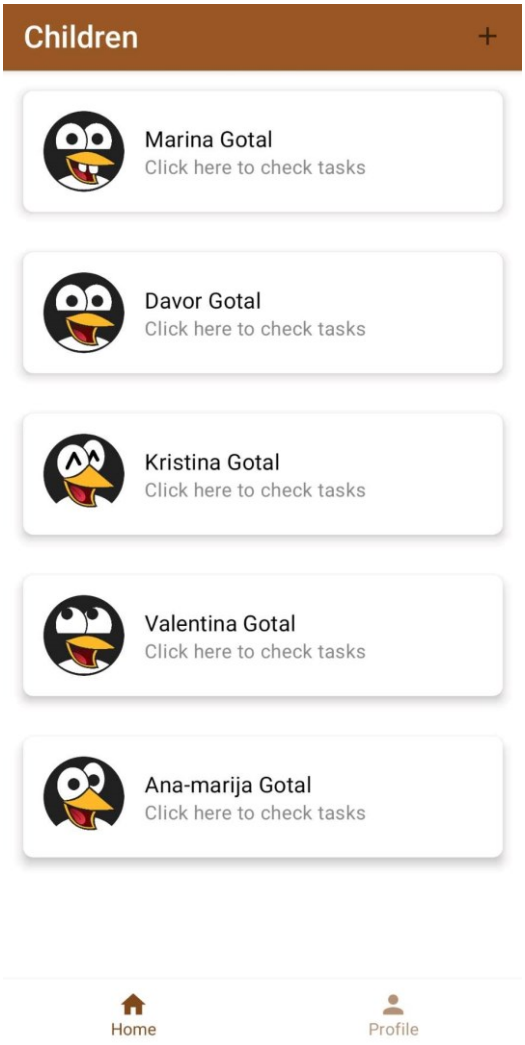
3.4. Početni korisnički zasloni

Nakon što se korisnik uspješno prijavi u aplikaciju ovisno o ulozi korisnika će se otvoriti odgovarajući početni zaslon. Za ulogu roditelj prikazat će se zaslon s listom djece i navigacijskom

trakom čije su opcije pregled djece i pregled profila. Djetetu će se nakon prijave prikazati zaslon s listom dostupnih zadataka što omogućava djetetu lako pregledavanje dostupnih zadataka. Dostupni zadaci su poredani prema datumu krajnjeg roka kako bi djeca znala koje zadatke prvo trebaju obaviti ako žele ostvariti nagradu za izvršen zadatak.

3.4.1. Prikaz zaslona korisnika uloge roditelja

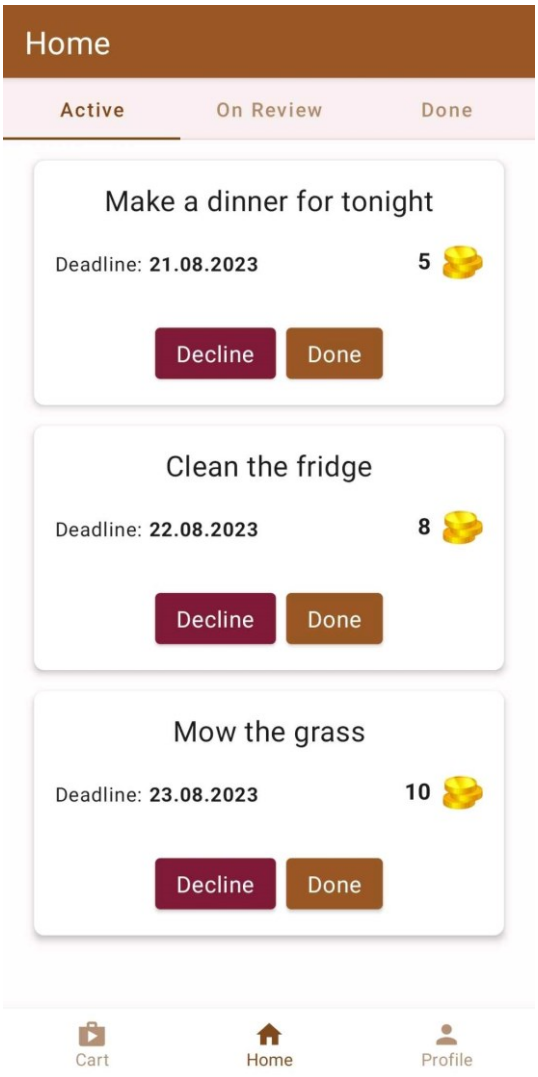
Prikazivanje djece na zaslonu je realizirano pomoću *Lazy Column* komponente. *Lazy Column* je *Compose* komponenta koja pruža intuitivan i organiziran način prikaza djece kako bi roditelji lako mogli pratiti aktivnosti djece. Kada korisnik uloge roditelj odabere dijete na zaslonu koji je prikazan na slici 3.4. aplikacija preusmjerava roditelja do novog zaslona koji jasno prikazuje zadatke, razvrstane prema njihovom statusu. Svaki korisnik može postaviti željenog avatara za svoju profilnu sliku koja će biti svima prikazana.



Slika 3.4. Prikaz djece

3.4.2. Prikaz zaslona korisnika uloge djeteta

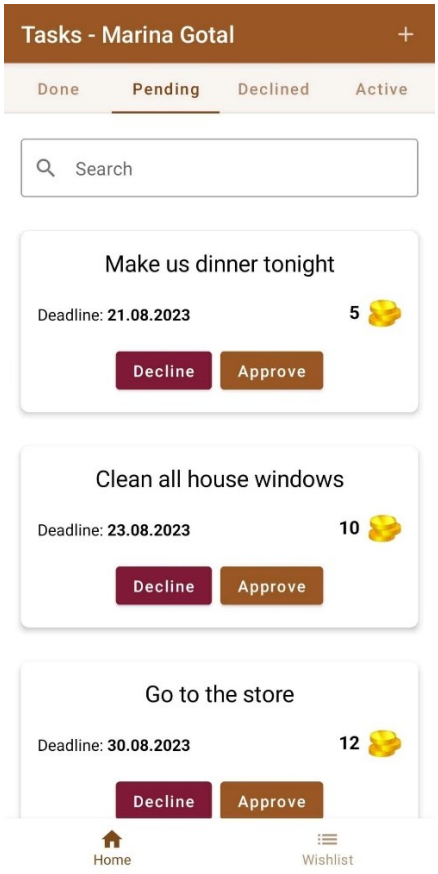
Početni zaslon korisnika uloge dijete otvara dostupne zadatke koje može izvršiti i poslati na pregled. Zadatke je moguće i odbiti ako ih dijete ne želi izvršiti kako bi roditelj pravovremeno znao da zadatak neće biti izvršen od strane njega te ga može dodijeliti nekom drugom. Zadaci su prikazani u *Lazy Column compose* komponenti s tri različita filtera koji filtriraju zadatke prema aktivnim zadacima koji su dostupni za izvršavanje, zadaci koji su na pregledu od strane roditelja i zadaci koji su završeni. Unutar svakog zadatka se nalazi i podatak o iznosu novčane nagrade koja će mu biti dodijeljena ako roditelj korisnika djeteta potvrdi da je zadatak uspješno obavljen. Ako je zadatak poslan na pregled, a roditelj ga vrati nazad na izvršavanjem jer nije uspješno obavljen, korisniku dijete će se prikazati zadatak s objašnjenjem zašto je zadatak neuspješno obavljen u kartici aktivnih zadataka kako bi zadatak popravio i poslao nazad na pregled roditelju.



Slika 3.5. Početni zaslon korisnika dijete

3.5. Upravljanje zadacima

Upravljanje zadacima obuhvaća dodavanje zadataka, brisanje zadataka, potvrdu zadataka, uređivanje zadataka, slanje nazad zadataka i odbijanje zadataka. Korisnik s ulogom roditelja ima mogućnost dodavati, brisati, odbijati i potvrđivati zadatke djeteta, dok korisnici s ulogom djeteta mogu pregledavati, odbiti ili obavljati dodijeljene zadatke radi ostvarivanja nagrada. Na slici 3.6 je prikazan raspored zadataka prema njegovom stanju. Zadaci koji su poslani na pregled roditelju od strane djeteta su prikazani na odjeljku zadataka koji su u tijeku, a imaju opcije potvrđivanje zadatka i odbijanje zadatka s porukom objašnjena.



Slika 3.6. Prikaz upravljanja zadacima

Status zadatka može biti:

1. Izvršen (engl. *Done*) – izvršeni zadata
2. U procesu prihvaćanja (engl. *Pending*)
3. Odbijen (engl. *Declined*)
4. Aktivan (engl. *Active*) - ako su zadaci aktivni, korisnik s ulogom roditelja može potvrditi ako su uspješno obavljeni ili vratiti djetetu s odgovarajućom porukom zadatak kako bi uspješno odradio u roku zadatak te obrisati ili dodati nove zadatke djetetu.

3.5.1. Dodavanje zadataka

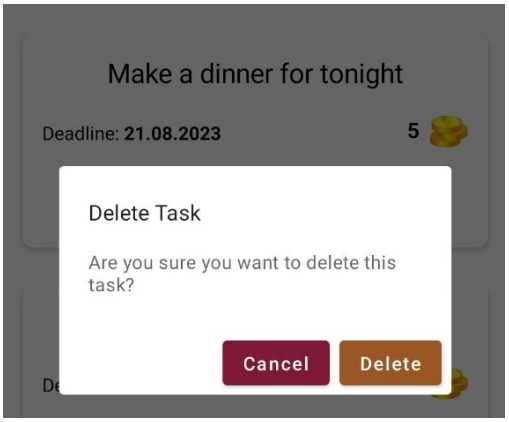
Jedna od ključnih funkcionalnosti aplikacije je mogućnost dodjela zadataka korisnicima čija je uloga dijete od strane korisnika čija je uloga roditelj. Za implementaciju funkcionalnosti dodavanja poslova korišten je kod prikazan u programskom kodu 3.4. Funkcija dodavanja zadataka kao argument ima objekt zadatak koji sadrži podatke o zadatku koji se dodaje. U bloku pokušaja (engl. *try*) prvo stvaramo referencu na dokument u *Firebase Firestore* bazi podataka te se dodjeljuje jedinstveni identifikator poslu. U slučaju pojavljivanja greške pokreće se blok za hvatanje iznimki (engl. *catch block*) koda i ispisuje se poruka o pogrešci.

```
fun addTask(task: Task) {
    viewModelScope.launch(Dispatchers.IO) {
        try {
            val documentRef = collectionRef.document()
            task.id = documentRef.id
            task.parentId = firebaseAuth.currentUser?.uid ?: ""
            documentRef.set(task).await()
            println("Task created successfully with ID: ${task.id}")
        } catch (e: Exception) {
            println("Error creating task: ${e.message}")
        }
    }
}
```

Programski kod 3.4. Programski kod za dodavanje poslova

3.5.2. Brisanje zadataka

Mogućnost brisanja zadataka je ključna kod organizacije obavljanja zadataka. Ako dodijeljeni zadatak više nije dostupan ili je dovršen od strane nekog drugog djeteta, korisnik čija je uloga roditelj može obrisati zadatak kako bi održavao ažuriranu listu zadataka. Ovo je posebno važno u situacijama kada se prioriteti ili uvjeti za obavljanje posla mijenjaju što je čest slučaju ako roditelj ima više djece te su zadaci krivo dodijeljeni ili dodijeljeni više korisnika. Na slici 3.7. je prikazan dijalog koji se otvara ako korisnik roditelj želi obrisati određeni zadatak.



Slika 3.7. Dijalog za brisanje zadataka

Funkcija brisanja zadataka koja je prikazana u programskom kodu 3.5. omogućuje roditelju brisanje određenog posla iz aplikacije. Kada roditelj pritisne gumb za brisanje, funkcija se poziva, a paralelni proces pokreće proces brisanja. Pozivom metode *delete()* nad dokumentom koji odgovara odabranom zadatku, taj zadatak se trajno uklanja iz baze podataka.

```
fun deleteTask(task: Task) {
    viewModelScope.launch {
        collectionRef.document(task.id).delete()
    }
}
```

Programski kod 3.5. Programski kod za brisanje poslova

Implementacija ove funkcionalnosti pruža roditelju fleksibilnost i kontrolu nad organizacijom obavljanja zadataka. Oni mogu brisati zadatke koji više nisu relevantni ili su dovršeni, čime održavaju preglednu i ažuriranu listu zadataka. Ova funkcionalnost također olakšava dodavanje novih zadataka i prilagođavanje rasporeda obaveza kako bi se odgovorilo na promjene i zahtjeve koji se javljaju tijekom vremena.

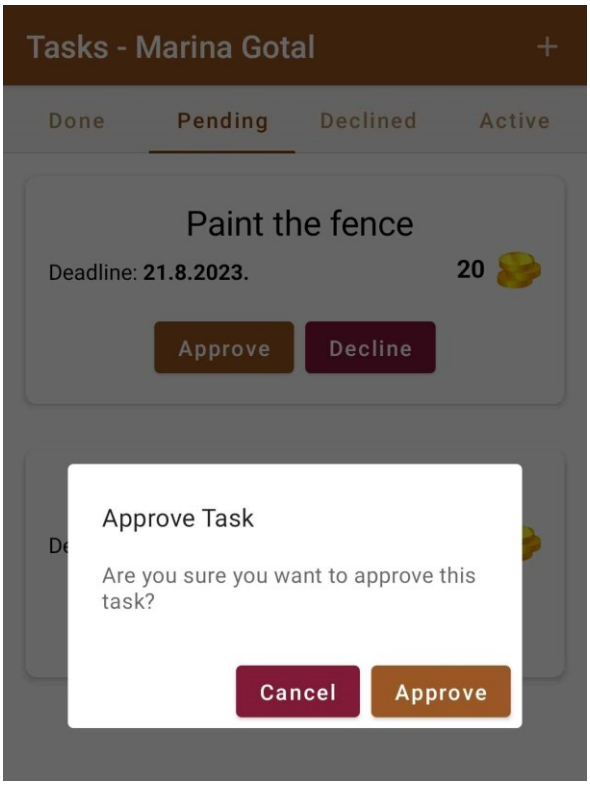
3.5.3. Potvrda zadataka

Potvrda zadatka omogućava roditeljima da potvrde završetak određenih zadataka kako bi se dodijelila adekvatna nagrada djetetu koje je obavilo zadatak. Funkcija kojom je omogućeno potvrđivanje zadataka prikazana je u programskom kodu 3.6.

```
fun approveTask(task: Task) {
    viewModelScope.launch {
        try {
            updateChildBalance(task)
            updateTaskApproval(task)
            removeTaskFromList(task)
        } catch (e: Exception) {
            println("Error approving task: ${e.message}")
        }
    }
}
```

Programski kod 3.6. Programski kod za potvrda poslova

U bloku pokušaja se pozivaju tri funkcije *updateChildBalance()*, *updateTaskApproval()* i *removeTaskFromList()* koje primaju za argument zadatak koji je uspješno obavljen. Funkcija *updateChildBalance()* pronalazi u bazi podataka dijete koje je obavilo zadataka te dodaje novčanu nagradu zadatka na djetetov virtualni račun. Nakon što se doda novčana nagrada funkcijom *updateTaskApproval()* se mijenja status zadatka na gotove te se zatim uklanja iz liste zadataka za pregled funkcijom *removeTaskFromList()*. Prikaz dijaloga potvrde zadataka je prikazan na slici 3.8.



Slika 3.8. Dijalog za potvrdu zadataka

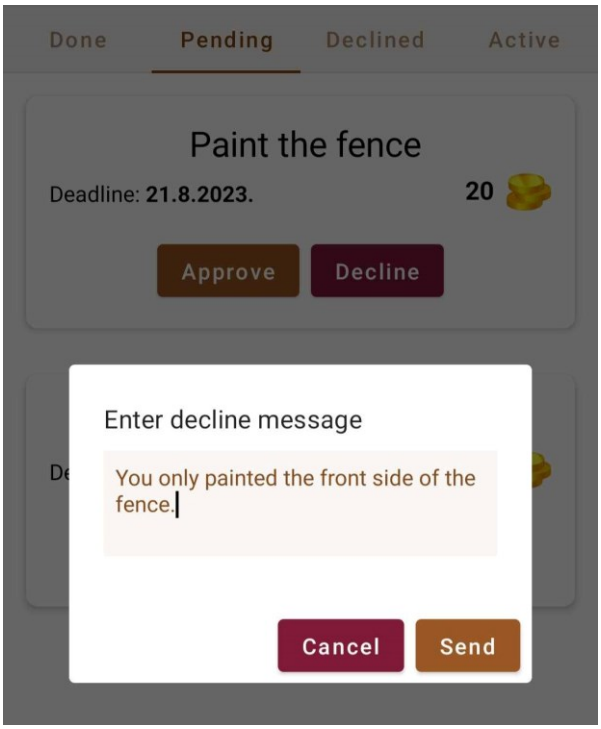
3.5.4. Slanje zadatka na ponovni pregled

Roditelj ima mogućnost ponovno poslati zadataka na ponovno izvršenje koji je dijete poslalo na pregled. U bazi se ažurira atribut koji predstavlja potvrđeno stanje od strane djeteta na neistinito i dodaje se poruka objašnjena zašto je zadatak vraćen kako bi dijete mogli ispraviti neuspješno obavljanje zadatka.

```
private suspend fun performDeclineTask(task: Task) {
    try {
        collectionRef.document(task.id)
            .update("declined", true)
            .await()
    } catch (e: Exception) {
        println("Error declining task: ${e.message}")
    }
}
```

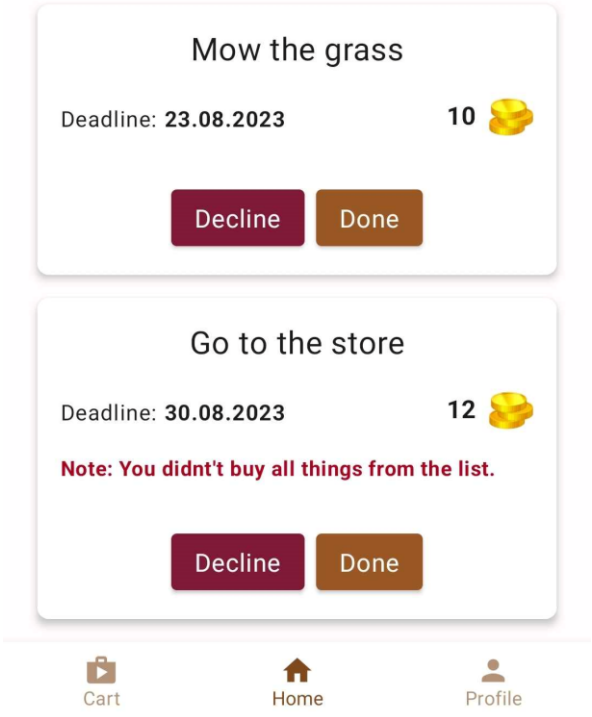
Programski kod 3.7. Programski kod za odbijanje zadataka

Prilikom klika na opciju odbijanja se otvara dijalog upozorenja koji prikazuju korisniku poruku o njegovoj odluci izvršenja željene akcije što je prikazano na slici 3.9. Unutar dijaloga upozorenja otvara se i tekstualno polje koje omogućava roditelju da unese poruku zašto je određeni zadatak odbijen kako bi dijete znalo pravovremeno ispraviti zadatak i ponovno ga poslati na pregled.



Slika 3.9: Slanje poruke odbijanja

Nakon što roditelj odbije neuspješno obavljen zadatak, djetetu će se unutar liste aktivnih zadataka prikazati odbijeni zadatak s razlogom zbog kojeg zadataka nije prihvaćen te kako ga može popraviti. Primjer poruke s razlogom odbijanja prikazan je na slici 3.10.



Slika 3.10. Prikaz neuspješnog zadatka na listi djetetovih zadataka

3.5.5. Uređivanje zadataka

Zadaci čiji status je aktivan imaju opciju uređivanja. Funkcija *editTask()* koja je prikazana na programskom kodu 3.8. prima kao argument podatke koje korisnik želi izmijeniti na postojećem zadatku. Ima mogućnost promijeniti naslov, iznos nagrade i rok izvršenja zadataka. Pozivom funkcije se pronalazi odabrani zadatak koji se želi urediti te se s trenutnim podacima popunjava dijalog kako bi roditelj lakše izmijenio podatke.

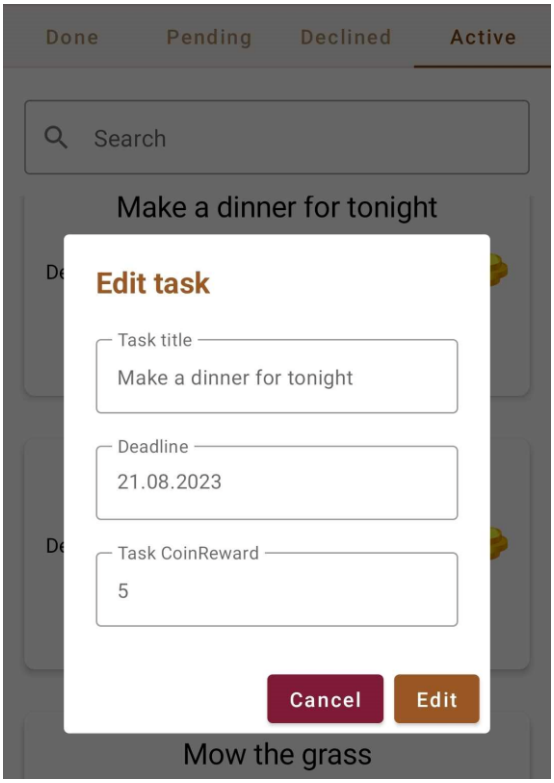
```
fun editTask(updatedTask: Task) {
    viewModelScope.launch(Dispatchers.IO) {
        try {
            val taskDocumentRef = collectionRef.document(updatedTask.id)
            taskDocumentRef.update(
                "title", updatedTask.title,
                "coinReward", updatedTask.coinReward,
                "deadline", updatedTask.deadline
            ).await()

            val clonedTaskList = ArrayList(_items.value)
            val index = _items.value.indexOfFirst { it.childId ==
updatedTask.childId }

            clonedTaskList [index] = clonedTaskList [index].copy(
                title = updatedTask.title,
                coinReward = updatedTask.coinReward,
                deadline = updatedTask.deadline
            )
            _items.emit(clonedTaskList)
            println("Task edited successfully: ${updatedTask.id}")
        } catch (e: Exception) {
            println("Error editing task: ${e.message}")
        }
    }
}
```

Programski kod 3.8. Programski kod za uređivanje zadataka

Unos krajnjeg roka izvršavanja zadatka ostvaren je *compose* komponentom *DatePicker* koja olakšava odabir datuma putem interaktivnog kalendara. Kalendar prikazuje označen današnji datum te onemogućava označavanje datuma koji je prošao. Postoji opcija i brojčanog unosa datuma koji ima provjeru ispravnosti unesenog formata kako datum ne bi bio pogrešno spremljen u bazu podataka.

Prilikom odabira uređivanja otvara se dijalog u kojem su prikazani podaci odabranog zadatka koji se mogu urediti. Roditelj može urediti naslov zadatka, datum izvršavanja zadatka i iznos novčane nagrade za izvršavanje zadatka. Korisnik roditelj ima opciju izaći iz dijaloga i ne promijeniti podatke klikom na dugme otkazivanja (engl. *Cancel*) ili klikom izvan područja otvorenog dijaloga. Promjene se spremaju u bazu prilikom klika na dugme ažuriranja (engl.

Update) koji poziva funkciju *updateTask()* i pronalazi zadatak u bazi i ažurira podatke tog zadataka. Zaslon uređivanja zadatka je prikazan na slici 3.6.



Slika 3.11. Zaslon za uređivanje zadatka

3.5.6. Pretraživanje zadataka

U aplikaciji je omogućeno pretraživanje zadataka po naslovu unosom riječi ili slova koja se nalaze u kreiranim zadacima. Funkcija prikazana u programskom kodu 3.9. prima za argument vrijednost unesenu u tražilicu i zatim izdvaja sve zadatke koje sadrže unesena slova bez obzira na velika i mala slova.

```
fun filterBySearch(value: String) {
    _items.value = tasks.filter { it.title.contains(value, true) }
}
```

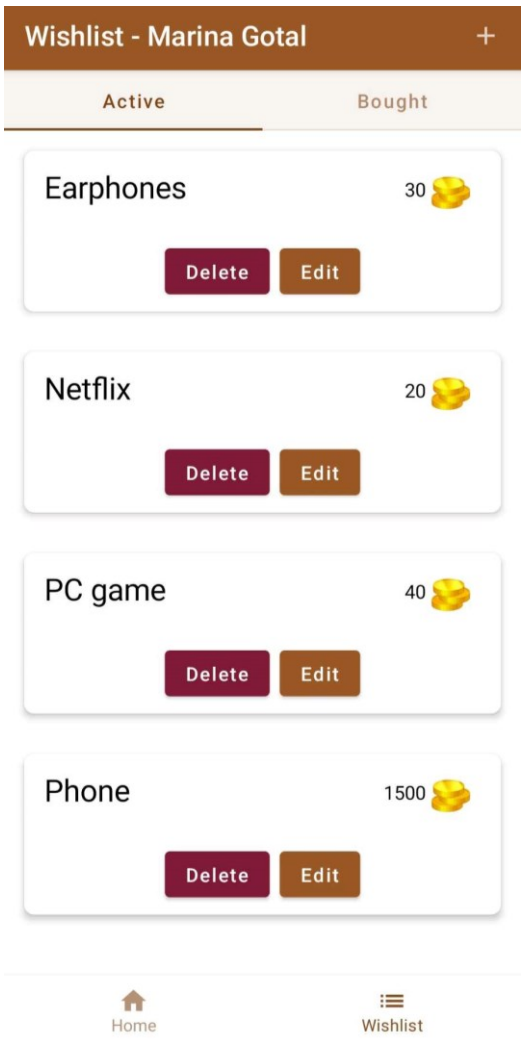
Programski kod 3.9. Pretraživanje zadataka

3.5.7. Automatsko ažuriranje zadataka

Svaki zadatak ima atribut koji govori do kada zadatak mora biti završen (engl. *deadline*). Ukoliko se zadatak ne izvrši do predviđenog datuma koji predstavlja krajnji rok, zadatak se označava odbijenim od strane djeteta te neće ići na pregled roditelju. Postavljanjem rokova za završavanje zadataka potiče se djecu na razvoj odgovornosti i poštivanje izvršavanja obaveza unutar vremenskih okvira.

3.6. Upravljanje željama

Upravljanje željama pruža korisnicima uloge roditelja jednostavno dodavanje, uređivanje i brisanje želja. Roditelj odabire što želi staviti kao cilj određenom djetetu koje može ostvariti izvršavajući zadatke, zajedno sa iznosom koji smatra da je adekvatan za određenu želju.



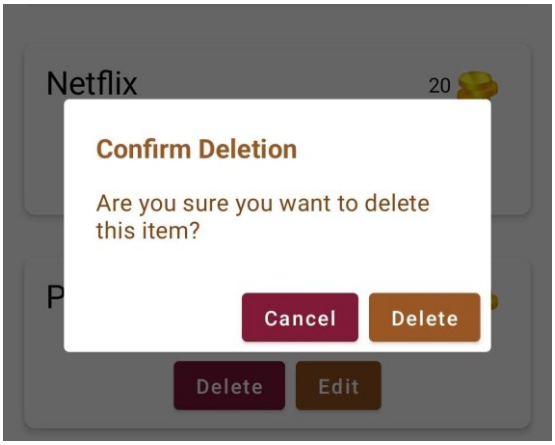
Slika 3.12. Prikaz zaslon želja

3.6.1. Dodavanje želja

Roditelj može dodavati želje u bazu podataka na sličan način na koji se dodaju zadaci određenom djetetu. Unošenjem naslova želje i iznosa novčane nagrade poziva se funkcija za dodavanje želje u bazu podataka koja za argument ima objekt želju koju dodaje u bazu podataka s podacima unesenim u formu koji je roditelj popunio te zajedno s tim podacima sprema i jedinstveni identifikator roditelja i djeteta za koje je želja namijenjena.

3.6.2. Brisanje želja

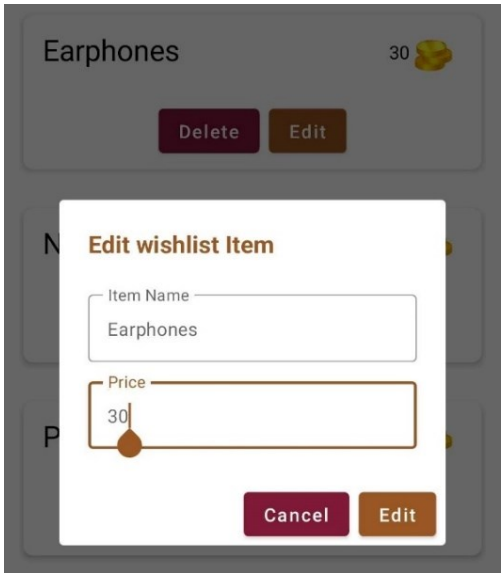
Želje koje nisu potvrđen od strane roditelja ili djeteta se mogu obrisati. Klikom na dugme brisanja (engl. *Delete*) se otvara dijalog potvrde o brisanju želje prikazanog na slici 3.12. koji ima opcije zatvaranja dijaloga i brisanje želje. Na odabir brisanja želje se poziva funkcija *DeleteWish()* koja odabranu želju pronalazi u bazi podataka i trajno ju uklanja.



Slika 3.13. Prikaz dijaloga za brisanje želje

3.6.3. Uređivanje želja

Uređivanje želja je realizirano kroz dijalog koji se otvara prilikom odabira opcije uređivanja. Unutar dijaloga prikazanog na slici 3.13. se nalaze dva polja za unos koji su popunjeni s trenutnim podacima zadatka. Nakon što roditelj promjeni vrijednosti u poljima za unos prilikom odabirom opcije uredi unutar dijaloga se poziva funkcija *EditWish()* koja pronalazi želju koju roditelj želi promijeniti i ažurira podatke u bazi podataka.



Slika 3.14. Prikaz uređivanja želje

4. ZAKLJUČAK

Završni rad prikazuje aplikaciju za uključivanje djece u svakodnevne aktivnosti putem dodjeljivanja zadataka. Aplikacija omogućuje roditeljima dodjeljivanje zadataka svojoj djeci te je na taj način djeci omogućeno da svojim trudom zarade nagrade za željene stvari poput novog mobitela, odlaska u kino, novi bicikl.

Nadogradnja aplikacije u budućnosti uključuje mogućnost kupovine željenih usluga i stvari kroz aplikaciju, stvarajući virtualnu košaricu s mogućim nagradama kao što su bicikl, odlazak u kino ili transakcija određenog iznosa na štednju. Nadogradnja vezano za izvršavanje zadataka bi potaknula suradnju i međusobno pomaganje i razvoj međuljudskih odnosa među djecom jer zajedničkim radom na zadacima bi postojala dodatna nagrada kako bi ih se motiviralo na zajednički rad i razvijanje timskog duha.

U nadogradnji aplikacije također postoji mogućnost komunikacije među korisnicima kako bi postojala mogućnost međusobnog dogovaranja oko izvršavanja zadataka, pružanje međusobne podrške i korisnih uputa za izvršavanje određenog zadatka ako korisnik nije siguran kako se određeni zadatak izvršava. Komunikacija unutar aplikacije bi doprinijela zainteresiranosti korištenja aplikacije i potaknula zajednički rad pospješujući komunikacijske i međuljudske vještine među djecom.

LITERATURA

[1] „Jetpack Compose UI App Development Toolkit - Android Developers“ [Online]. Dostupno na: https://developer.android.com/jetpack/compose?fbclid=IwAR3K0IoiblFcLo0HqkKWOLMNFLMTVz4xN4JfLgrXyo5I-5afU-Gjh2VWim8. [12. lipnja 2023.].

[2] N. Smyth, *Android Studio 4.1 Development Essentials - Kotlin Edition*. Payload Media, 2020.

[3] P. Sommerhoff, *Kotlin for Android App Development*. Addison-Wesley Professional, 2018.

[4] „Top 5 Best Android App Development Languages for 2023“ [Online]. Dostupno na: https://www.designveloper.com/blog/android-app-development-languages/?fbclid=IwAR1PzWpYhRmpDwtKQ-jfUG0f4FTksp5OFHMFUJ3xIfQgm-AjXtQmaUq0CsA. [13. lipnja 2023.].

[5] „Kotlin for Android | Kotlin Documentation“ [Online]. Dostupno na: https://kotlinlang.org/docs/android-overview.html [15. lipnja 2023.].

[6] N. Smyth, *Jetpack Compose 1.3 Essentials: Developing Android Apps with Jetpack Compose 1.3, Android Studio, and Kotlin*. eBookFrenzy, 2023.

[7] „Preview your UI with Composable previews | Jetpack Compose“, *Android Developers*, [Online]. Dostupno na: https://developer.android.com/jetpack/compose/tooling/previews. [16. lipnja 2023.].

[8] „Firebase | Google’s Mobile and Web App Development Platform“, *Firebase*, [Online]. Dostupno na: https://firebase.google.com/. [17. lipnja 2023.].

SAŽETAK

Cilj ovog završnog rada bio je stvoriti mobilnu aplikaciju za aktivno uključivanje mladih u svakodnevne aktivnosti. Aplikacija omogućava kreiranje, dodjeljivanje i upravljanje zadataka koje roditelj dodjeljuje djeci. Teorijski dio rada opisuje tehnologije korištene u izradi aplikacije, te opisuje slijed razvoja aplikacije. Praktični dio rada sastoji se od izrade same aplikacije. Izgled i struktura mobilne aplikacije rađeni su pomoću Jetpack Compose alata za izradu korisničkog sučelja. Funkcionalni dio aplikacije napravljen je u Kotlin programskom jeziku. Za izradu baze podataka korištena je Firebase platforma.

Ključne riječi: Android, baza podataka, djeca, Jetpack Compose, Kotlin, mobilna aplikacija.

ABSTRACT

Mobile Application for Active Involvement of Children in Everyday Activities

The objective of this final project was to develop a mobile application aimed at actively engaging young people in their daily activities. The application allows for the creation, assignment, and management of tasks delegated by parents to their children. The theoretical section of this paper provides an overview of the technologies employed in application development and outlines the subsequent phases of application development. The practical part of the final project consists of creating the application itself. The layout and structure of the mobile application were constructed using the Jetpack Compose UI builder, while the functional aspect of the application was implemented in the Kotlin programming language. The Firebase platform was utilized to establish and manage the database.

Keywords: Android, children, databse, Jetpack Compose, Kotlin, mobile application.

ŽIVOTOPIS

Davor Gotal rođen je 20. siječnja 2002. godine u Osijeku, Hrvatska. Prvih dva razreda osnovne škole pohađa u osnovnoj školi Mladost u Osijeku, nakon čega se seli u Bilje i tamo završava svoje osnovnoškolsko obrazovanje u OS Bilje. Nakon osnovne škole upisuje za Tehničara za računalstvo u Elektrotehničkoj i prometnoj školi, Osijek. 2020. godine dobiva izravan upis na preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija te ga upisuje iste godine.

PRILOZI

4.1. Izvorni kod aplikacije

Dostupan na: https://github.com/dgotal/help_and_earn.git