Izrada mobilne aplikacije "To do"

Ivanković, Vedran

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Vedran Ivanković

Izrada mobilne aplikacije „To do“

Završni rad

Mentor: izv. prof. dr. sc. Marija Brkić Bakarić

Rijeka, srpanj 2024.

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija
uniri

Rijeka, 3.6.2024.

Zadatak za završni rad

Pristupnik/ica: Vedran Ivanković

Naziv završnog rada: Izrada mobile aplikacije "To do"

Naziv završnog rada na engleskom jeziku: Development of the mobile application "To Do"

Sadržaj zadatka:
U okviru ovog rada izradit će se mobilna aplikacije za unošenje bilješki na vlastitu to-do listu korištenjem softverskog razvojnog kompleta (engl. *software development kit*, skraćeno SDK) Flutter. Cilj mobilne aplikacije „you to-do" je omogućiti korisniku kreiranje korisničkog računa putem unosa svoje email adrese i željene lozinke, prijavu u aplikaciju te dodavanje bilješki, kao i izvršavanja ostalih operacija poput označavanja bilješki kao dovršenih, pretraživanja bilješki, brisanja određene bilješke, sortiranja i filtriranja.

Mentor/ica
Izv. prof. dr. sc. Marija Brkić Bakarić

Voditelj za završne radove
Izv. prof. dr. sc. Miran Pobar

Zadatak preuzet: 3.6.2024.

(potpis pristupnika/ice)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

Ovaj rad prikazuje izradu mobilne aplikacije za unošenje bilješki na vlastitu to-do listu korištenjem softverskog razvojnog kompleta (engl. s*oftware development kit*, skraćeno SDK) Flutter.

Cilj ove mobilne aplikacije je omogućiti korisniku kreiranje korisničkog računa putem unosa svoje email adrese i željene lozinke, prijavu u aplikaciju te dodavanje bilješki, kao i izvršavanja ostalih operacija poput označavanja bilješki kao dovršenih, pretraživanja bilješki, brisanja određene bilješke, sortiranja i filtriranja.

Na početku rada ukratko je opisan Flutter SDK, a potom je u nekoliko logičkih cjelina kroz prikaz slika zaslona i odsječaka koda dan cjelokupan proces izrade aplikacije.

Aplikacija se sastoji od početnog zaslona koji prikazuje logo aplikacije, poruku dobrodošlice, tipku za registraciju, koja je namijenjena za nove korisnike, te tipku za prijavu, koja je namijenjena za već postojeće korisnike. Zasloni za registraciju i prijavu se sastoje od polja za unos email adrese i lozinke, dok se na zaslonu za prijavu još nalazi i mogućnost promjene lozinke u slučaju da ju je korisnik zaboravio. Kada se korisnik uspješno registrira, prikazuje se zaslon o uspješnoj registraciji i poruka kako može početi koristiti aplikaciju. Nakon toga se prikazuje glavni zaslon aplikacije na kojemu se vrše sve operacije vezane uz upravljanje bilješkama. Navedeni zaslon također prilikom svakog sljedećeg korištenja aplikacije postaje početnim (osim ako se korisnik ne odjavi prije izlaza iz aplikacije) kako bi se olakšalo i ubrzalo korištenje aplikacije.

Ključne riječi: Flutter; Google Firebase; mobilna aplikacija; bilješke; unos; widgeti

SADRŽAJ

1. Uvod

Mobilne aplikacije neizostavan su dio ljudske svakodnevice. Posve je uobičajeno koristiti se aplikacijama za pretraživanje interneta, slušanje glazbe, gledanje filmova i videozapisa, slanje poruka i sličnim. U posljednjih desetak godina razvile su se i mobilne aplikacije koje se bave praktičnim aspektima ljudskih života u svrhu poboljšanja kvalitete života i olakšavanja izvršavanja svakodnevnih zadataka. Tako, primjerice, postoje aplikacije za internet bankarstvo koje nude upravljanje vlastitim financijama i olakšavaju plaćanje računa, aplikacije za trgovinu koje nude naručivanje proizvoda iz udobnosti vlastitog doma, aplikacije vezane uz zdravlje koje sadrže informacije o pacijentu, fizičkoj aktivnosti itd.

Postoje mnogi razvojni okviri i SDK-ovi za razvoj desktop i mobilnih aplikacija, a jedan od njih je i Flutter, koji će detaljnije biti predstavljen u ovome radu te koji će biti korišten za izradu mobilne aplikacije pod nazivom „you to-do“, koja omogućuje unošenje i upravljanje bilješkama na vlastitoj to-do listi. Aplikacija je namijenjena olakšavanju svakodnevice korisnicima, omogućujući korisniku brzo i jednostavno praćenje vlastitih obaveza putem unosa bilješki.

U prvom poglavlju dan je kratki uvod u Flutter. U drugome poglavlju rada prikazan je dizajn korisničkog sučelja, a u trećem implementacija podijeljena u potpoglavlja koja prikazuju implementaciju pojedinih zaslona, autentifikacije, podržanih operacija na listi bilješki te korisničkog unosa. Na kraju rada nalazi se zaključak.

2. Flutter

Flutter je komplet za razvoj softvera kojeg je razvio Google. Objavljen je 2017. godine. Otvorenog je koda (engl. o*pen-source*) te je namijenjen izradi desktop, web i mobilnih aplikacija. Flutter aplikacije se programiraju u Dart programskom kodu, koji je također razvijen od strane Google-a. Jedna od najvećih prednosti Fluttera je jednostavnost i lakoća korištenja. Programiranjem u Flutteru uvelike se smanjuje količina posla, s obzirom da se za sve platforme koristi isti kod.

Također, jedna od važnih prednosti je i *hot reload* funkcionalnost. *Hot reload* omogućuje osvježavanje aplikacije na uređaju ili emulatoru tijekom programiranja, čime se u aplikaciji odmah prikazuju sve promjene koje su napravljene u kodu. Flutter koristi koncept *widgeta* za izgradnju korisničkog sučelja, pri čemu je sve, od gumbova, polja za unos pa do složenih dizajna zaslona, napravljeno korištenjem *widgeta*. Ovakav pristup omogućuje veliku fleksibilnost i prilagodljivost prilikom dizajniranja aplikacija.

2.1. Navigacija

Jedan od ključnih aspekata svake aplikacije je navigacija, odnosno kretanje kroz zaslone unutar aplikacije. Flutter nudi nekoliko načina implementacije navigacije. Prvi i najjednostavniji način implementiranja navigacije je korištenjem `Navigator` *widgeta*, koji ekrane aplikacije sprema u stog. `Navigator` *widget* ima dvije glavne operacije vezane uz stog: push i pop.

Prva operacija, push, koristi se za dodavanje novog zaslona na vrh stoga. Ova operacija omogućuje korisniku da se prebaci na novi zaslon, dok se prethodni zaslon zadržava u pozadini.

```
TextButton(
  onPressed: () {
    Navigator.push(context,
        MaterialPageRoute(builder: (context) => const LoginPage()));
  }
);
```

Primjerice, u gore navedenom primjeru, push operacija koristi `MaterialPageRoute` klasu koja poziva *widget* `LoginPage`, koji korisnika vodi na zaslon za prijavu.

S druge strane, operacija pop uklanja trenutno prikazani zaslon s vrha stoga i vraća korisnika na prethodni zaslon. Primjerice, ako se želi napraviti da se korisnika vrati na prethodni zaslon pritiskom na neku tipku, to se može učiniti tako da se unutar `onPressed` metode pozove navedena linija koda u sljedećem primjeru:

```
TextButton(
  onPressed: () {
    Navigator.pop(context);
  }
);
```

Također, korištenjem pop operacije nad `SystemNavigator` *widgetom*, osigurava se da se pritiskom tipke natrag na mobilnom uređaju (ili pokretom unazad, ovisno o tome koristi li korisnik navigaciju tipkama ili pokretima na svom uređaju) izađe iz aplikacije. Sljedeći isječak koda prikazuje kako se to može implementirati unutar *widgeta* `PopScope`.

```
PopScope(
  canPop: true,
  onPopInvoked: (didPop) {
    if (didPop) {
      SystemNavigator.pop();
    }
);
```

Korištenje pop operacije je vrlo jednostavan i efektivan način da se izbjegnu neželjeni procesi unutar aplikacije, kao, na primjer, vraćanje na ekran za prijavu u aplikaciju nakon što se korisnik već prijavio. Obje navedene operacije omogućuju osnovnu navigaciju unutar aplikacije te su dovoljno jednostavne za implementaciju u većini aplikacija.

Ako se razvija nešto složenija aplikacija koja će, osim na mobilnim uređajima, biti dostupna i na web preglednicima, navigaciju se može implementirati i korištenjem imenovanih ruta (engl. *named routes*). Ovaj pristup omogućuje definiranje svih mogućih ruta aplikacije, što olakšava održavanje i upravljanje navigacijskom logikom, prvenstveno u složenijim aplikacijama.

```
MaterialApp(
  routes: {
    '/': (context) => HomeScreen(),
    '/details': (context) => DetailsScreen(),
  });
```

Definiranje imenovanih ruta obavlja se unutar `MaterialApp` *widgeta* unutar parametra routes, gdje se svakoj ruti dodjeljuje jedinstveni naziv i *widget* zaslona na koji će ta ruta prebaciti korisnika. U gore navedenom primjeru navedene su dvije rute: ruta koja vodi na početni zaslon te ruta koja vodi na zaslon s detaljima. Iako je ovo jedan od načina implementacije navigacije, Flutter dokumentacija navodi kako se ipak ne preporuča korištenje istog za većinu aplikacija. Kao razlozi navodi se problem do kojeg dolazi prilikom primanja nove duboke veze (engl. *deep link*, odnosi se na otvaranje Flutter aplikacije iz neke druge aplikacije ili preko poveznice), kada Flutter gura novu rutu na stog `Navigator` *widgeta*, neovisno o tome gdje se korisnik trenutno nalazi. Uz ovaj nedostatak, Flutter u aplikacijama koje koriste imenovane rute, ne podržava korištenje gumba prema naprijed u web preglednicima, što dovodi do neočekivanog ponašanja aplikacije.

U ovom radu se koriste i `Navigator` i `SystemNavigator` *widgeti* kako bi se implementiralo vraćanje na prethodni zaslon unutar aplikacije te izlaz iz aplikacije.

2.2. Upravljanje stanjem

Stanje u Flutteru je skup podataka koji definira izgled i ponašanje *widgeta* prilikom korištenja i interakcije s aplikacijom. Upravljanje stanjem se u Flutteru koristi za održavanje i ažuriranje vizualnih elemenata unutar sučelja aplikacije. Na taj način se osigurava da se u korisničkom sučelju aplikacije prikazuju točni i ažurni podaci te promjene tijekom korisničke interakcije s aplikacijom (npr. unos korisničkih podataka za prijavu u aplikaciju). Flutter nudi dva osnovna tipa *widgeta* o kojima ovisi hoće li trenutni ekran u svom sučelju uvijek biti isti ili će sadržavati promjene na temelju korisničke interakcije, a to su Stateless i Stateful *widgeti*.

Stateless *widget*, kao što i ime govori, je *widget* koji nema stanje te predstavlja komponente aplikacije koje se ne mijenjaju, odnosno koje su prilikom svakog korištenja aplikacije jednake.

Koriste se za prikaz statičkih, nepromjenjivih elemenata kao što su slike, tekstovi, gumbovi itd. Najjednostavniji primjer primjene Stateless *widgeta* je, primjerice, na početnoj stranici aplikacije koja sadrži neku pozadinsku sliku, tekst dobrodošlice te gumbove za registraciju i prijavu.

S druge strane, Stateful *widget* je *widget* koji omogućuje promjene i upravljanje stanjem unutar Flutter aplikacije. Navedeni *widget* ima sposobnost promjene svog unutarnjeg stanja i promjene svog izgleda ovisno o interakciji korisnika s aplikacijom. Sadrži dva odvojena dijela: *widget* i njegov povezani *state* objekt. *Widget* se koristi za definiranje strukture i izgleda elemenata korisničkog sučelja (engl. *user interface*, skraćeno UI), dok se *state* objekt koristi za pohranu i manipulaciju stanjem tog *widgeta*. Kada se stanje unutar *widgeta* promijeni, Flutter automatski ponovno renderira *widget* kako bi prikazao nove podatke ili promjene u interakciji s korisnikom. Stateful *widgeti* su zbog navedenih karakteristika vrlo korisni kod implementacije raznih značajki unutar aplikacije, kao što su unos i pohrana podataka za registraciju/prijavu, prikaz kontekstnog izbornika (engl. *popup menu*) prilikom pritiska na gumb, promjena prikazanog broja stavki u košarici dodavanjem artikala u košaricu (u aplikacijama za trgovinu) i sl.

Stateful *widgeti* također imaju i metode koje su vezane uz upravljanje i ažuriranje stanja. Neke od najvažnijih metoda su:

- `initState()`
- `dispose()`
- `build()`
- `setState()`

Prva metoda `initState` se poziva jednom i to prilikom stvaranja *state* objekta. Služi za inicijalizaciju i postavljanje početnih vrijednosti željenih varijabli koje su definirane unutar Stateful *widgeta*.

Metoda `dispose` je usko povezana s metodom `initState` iz razloga što se pozivanjem te metode *state* objekt trajno uklanja iz *widget* stabla. Koristi se za procese poput primjerice odjavljivanja kontrolera, a sve u svrhu izbjegavanja curenja memorije (engl. *memory leaking*).

Metoda `build` je metoda koja se koristi za „izgradnju“ *widgeta*. Poziva se svaki put kada se treba prikazati *widget*, odnosno kada se *widget* ubacuje u stablo *widgeta* u određenom `BuildContext` objektu koji se koristi za praćenje i lociranje pozicije svakog *widgeta* u stablu.

Posljednja od navedenih metoda je `setState`. Ova metoda je jedan od najjednostavnijih načina ažuriranja stanja Stateful *widgeta*. Koristi se za ponovnu „izgradnju“ Stateful *widgeta* prilikom ažuriranja i promjene elemenata u korisničkom sučelju.

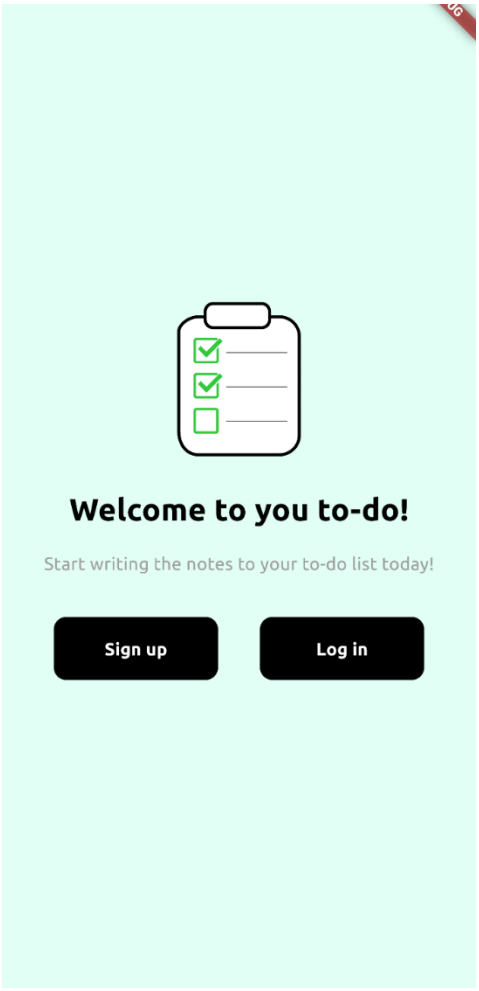
Prilikom izrade mobilne aplikacije „you to-do“ kroz 12 datoteka definirano je 6 Stateful i 6 Stateless *widgeta*.

3. Dizajn korisničkog sučelja „you to-do” aplikacije

U ovom poglavlju rada će biti prikazan i pojašnjen proces izrade aplikacije pod imenom „you to-do“. Aplikacija „you to-do“ je aplikacija namijenjena za dodavanje bilješki na vlastitu to-do listu. Osim unosa i dodavanja bilješki, aplikacija nudi još brojne funkcionalnosti. Bilješke se mogu označavati kao dovršene te se mogu i brisati, neovisno o tome jesu li dovršene ili ne. Osim toga bilješke se mogu pretraživati prema tekstu koji sadrže, mogu se sortirati prema najnovije dodanim, najstarije dodanim te uzlazno i silazno prema abecednom redoslijedu. Također, bilješke se mogu filtrirati i prema oznaci (engl. *tag*) koja je dodijeljena toj bilješci prilikom unosa. No, kako bi se sama aplikacija uopće mogla koristiti, potrebno je prvo napraviti korisnički račun i verificirati ga. Nakon toga račun će se moći nesmetano koristiti za prijavu u aplikaciju, a prilikom svake sljedeće prijave u aplikaciju korisnik će automatski ostati prijavljen, osim ako se nije odjavio prije izlaska iz aplikacije. Za korištenje aplikacije, i kod registracije/prijave, kao i kod upravljanja bilješkama, potrebno je da je uređaj povezan na Internet, s obzirom da je aplikacija povezana na Google Firebase.

Slijedi detaljno pojašnjenje dizajna korisničkog sučelja aplikacija, uz odgovarajuće snimke zaslona za svaki zaslon koji postoji u aplikaciji, uključujući i snimke zaslona koje prikazuju kako su u aplikaciji prikazane poruke koje korisnika obavještavaju o greškama prilikom registracije/prijave, unosa bilješki i sl.

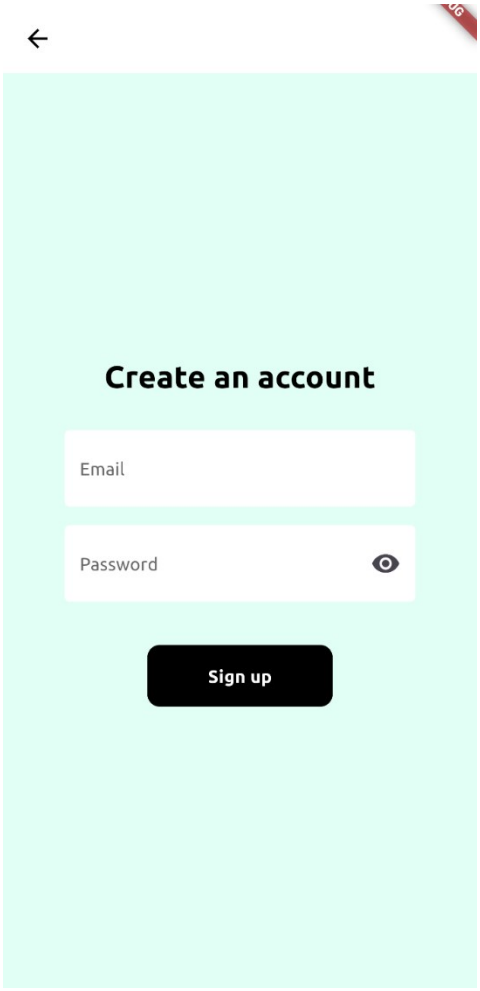
3.1. Početni zaslon



Slika 1. Korisničko sučelje početnog zaslona

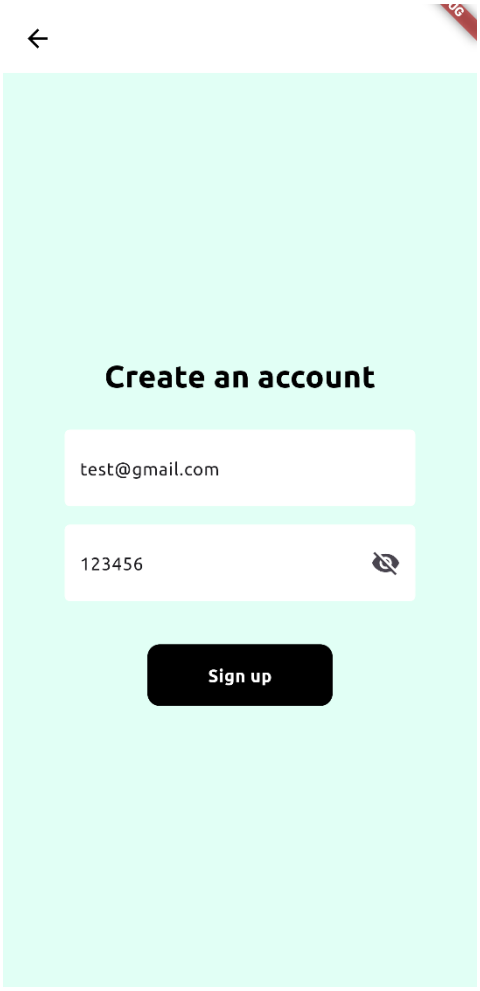
Početni zaslon ove aplikacije je prilično jednostavan. Definirana je blaga, pastelna boja pozadine koja je mješavina između plave i zelene boje (heksadekadska vrijednost: #E1FFF5) i koja se koristi na svim zaslonima aplikacije. Na sredini zaslona se zatim nalazi logotip aplikacije, ispod kojeg se nalazi poruka dobrodošlice i još jedna manja poruka. Tekstovi na ovom zaslonu i na svim ostalim zaslonima koriste font Ubuntu. Nakon toga se nalaze dva najbitnija elementa na ovom zaslonu, a to su tipka za registraciju (Sign up) i tipka za prijavu (Log in). Tipka za registraciju vodi na zaslon gdje se novi korisnici mogu registrirati, dok tipka za prijavu vodi na zaslon gdje se već registrirani korisnici mogu prijaviti u aplikaciju.

3.2. Zaslon za registraciju



Slika 2. Korisničko sučelje zaslona za registraciju

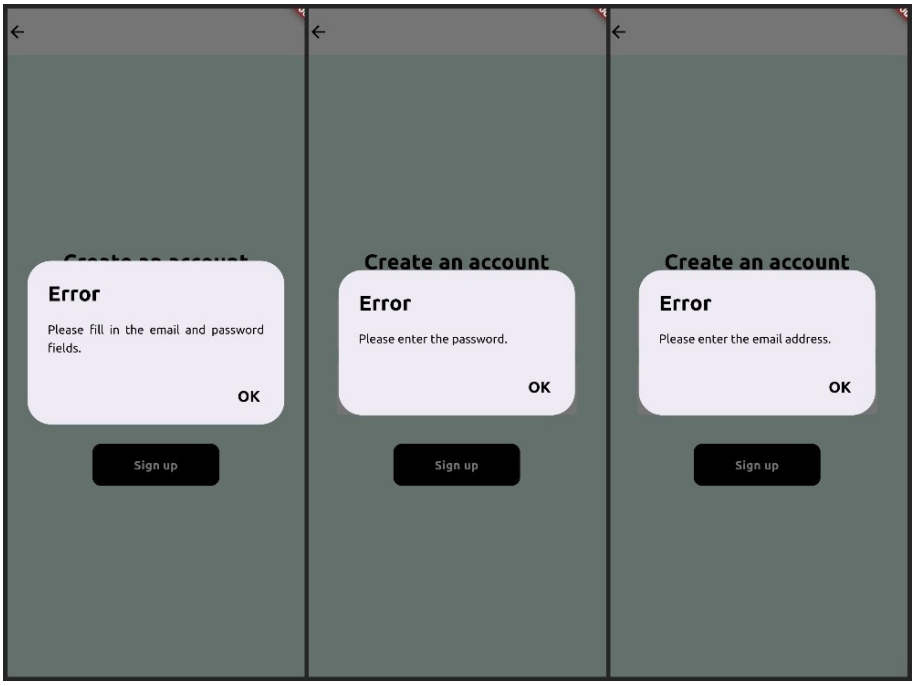
Za razliku od početnog zaslona, zaslon za registraciju sadrži alatnu traku. Alatna traka je bijele boje s tipkom za povratak unatrag, kako bi se korisniku omogućio povratak na početni zaslon. Na sredini zaslona nalazi se tekst „Create an account“ koji korisniku sugerira da se na ovom zaslonu vrši registracija za nove korisnike, a ispod tog teksta se nalaze polja za unos email adrese i lozinke, kao i tipka za registraciju.



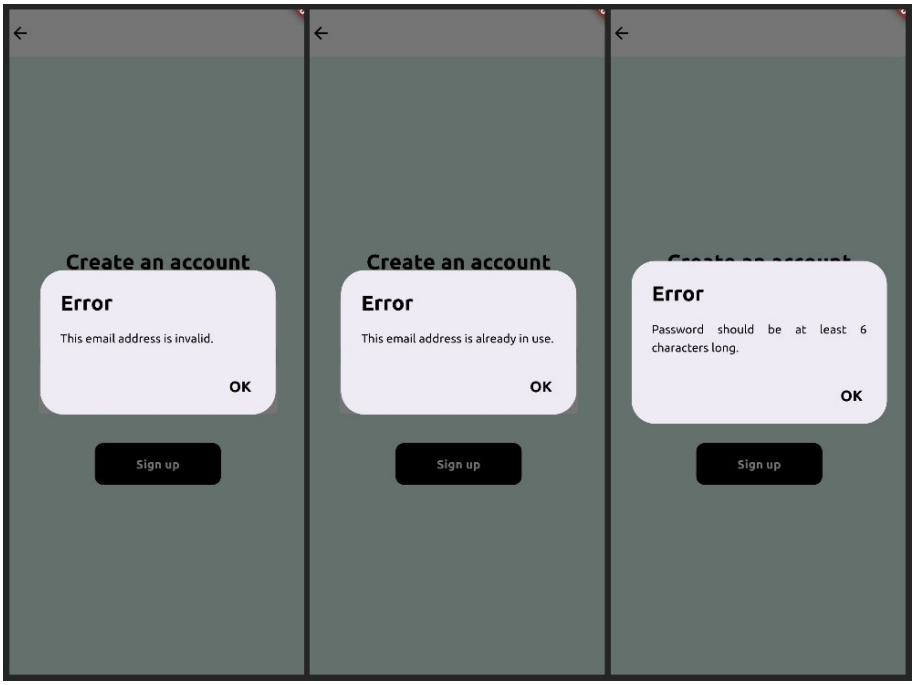
Slika 3. Primjer unosa i rada tipke za prikaz lozinke

Na slici iznad prikazan je primjer unosa email adrese i lozinke te prikaz kako se ikona tipke za prikaz lozinke promijeni kada ju korisnik stisne i lozinka se tada prikaže. Ponovnim pritiskom na tipku, prikaz lozinke se vraća u prvotno stanje, odnosno lozinka se prikazuje šifrirano.

Na ovom zaslonu postoji mogućnost da korisnik unese neke netočne i nepravilne vrijednosti koje dovode do grešaka prilikom pokušaja registracije u aplikaciju.



Slika 4. Poruke o greškama prilikom unosa - prvi dio



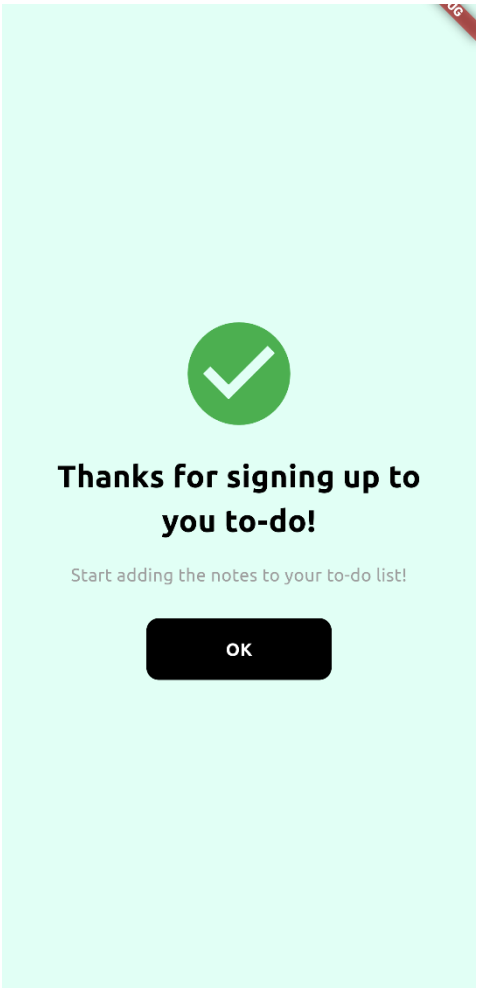
Slika 5. Poruke o greškama prilikom unosa - drugi dio

Na slikama iznad nalaze se sve greške do kojih dolazi prilikom registracije i koje uzrokuje korisnik svojim unosom. Na slici 4 prikazane su slike zaslona triju prozora s porukama o nedostatku unosa. Lijeva slika prikazuje poruku kako se moli korisnika da popuni oba polja za unos prije nego stisne tipku za registraciju. Srednja slika prikazuje kako nije unesena lozinka, dok desna slika zaslona prikazuje kako nije unesena email adresa.

Na slici 5 prikazane su slike zaslona grešaka do kojih dolazi iako su ispunjena oba polja za unos korisničkih podataka. Lijeva slika prikazuje poruku kako je unesena email adresa nevažeća, što sugerira da nije unesen točan format email adrese (npr. fali znak @). Srednja

slika prikazuje da se unesena email adresa već koristi, odnosno da već postoji korisnički račun povezan uz tu email adresu. Zadnja slika prikazuje poruku kako je unesena prekratka lozinka, te kako mora biti dugačka barem šest znakova.

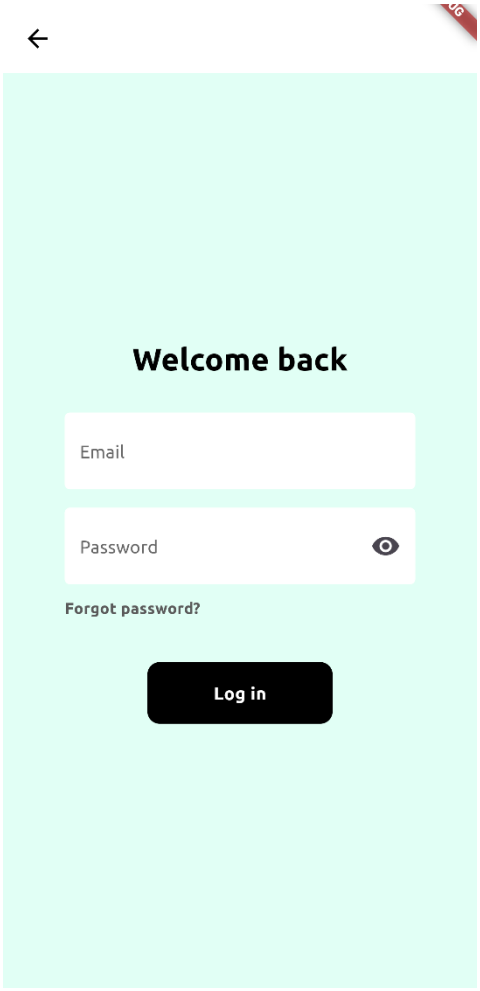
3.3. Zaslon o uspješnoj registraciji



Slika 6. Zaslon s porukom o uspješnoj registraciji

Nakon što se korisnik uspješno registrirao i verificirao svoju email adresu, prikazuje se zaslon o uspješnoj registraciji korisničkog računa za ovu aplikaciju. Na sredini zaslona prikazana je ikona kvačice koja je zelene boje, pozdravna poruka te manja poruka koja obavještava korisnika da može započeti dodavati bilješke na svoju to-do listu. Ispod svega toga nalazi se tipka „OK“, koja vodi korisnika na glavni zaslon aplikacije gdje se upravlja bilješkama na to-do listi. Na ovome zaslonu se ne nalazi alatna traka s tipkom za povratak unatrag iz razloga što bi to moglo dovesti do neočekivanog ponašanja aplikacije, dok se ovako korisnika potiče da nastavi dalje prema glavnom zaslonu aplikacije.

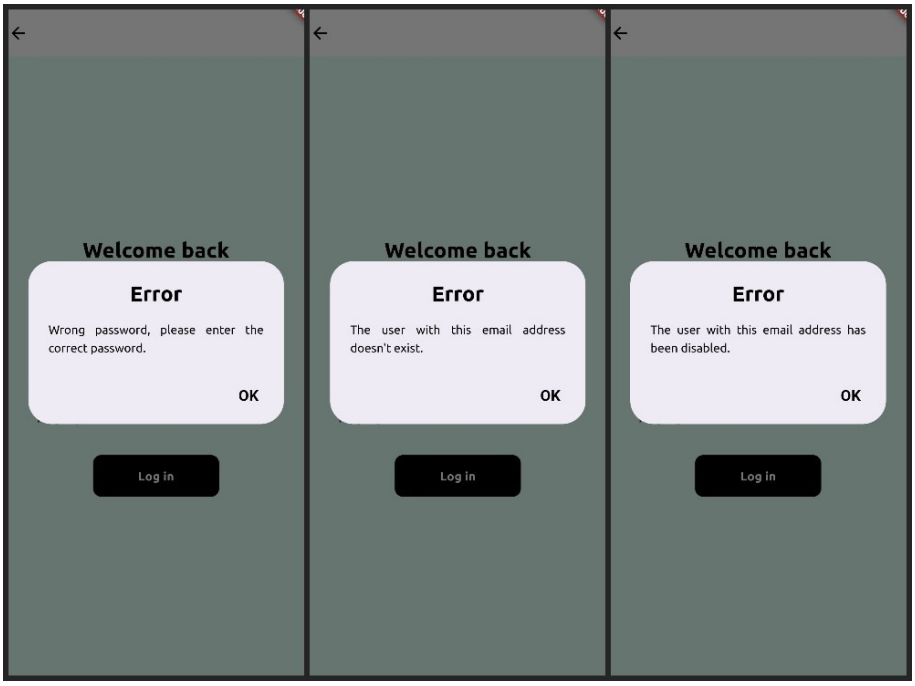
3.4. Zaslon za prijavu



Slika 7. Korisničko sučelje zaslona za prijavu

Zaslon za prijavu u aplikaciju je vrlo sličan zaslonu za registraciju. Unos email adrese i lozinke izgleda i funkcionira jednako kao i kod registracije. Ipak, ovaj zaslon ima tekst „Welcome back“ koji pozdravlja korisnika, te tipku „Log in“ koja korisnika vodi na glavni zaslon aplikacije. Uz to, na ovom zaslonu se ispod polja za unos lozinke nalazi i tekst „Forgot password?“ na koji se može kliknuti i koji nudi korisniku mogućnost resetiranja lozinke na sljedećem ekranu.

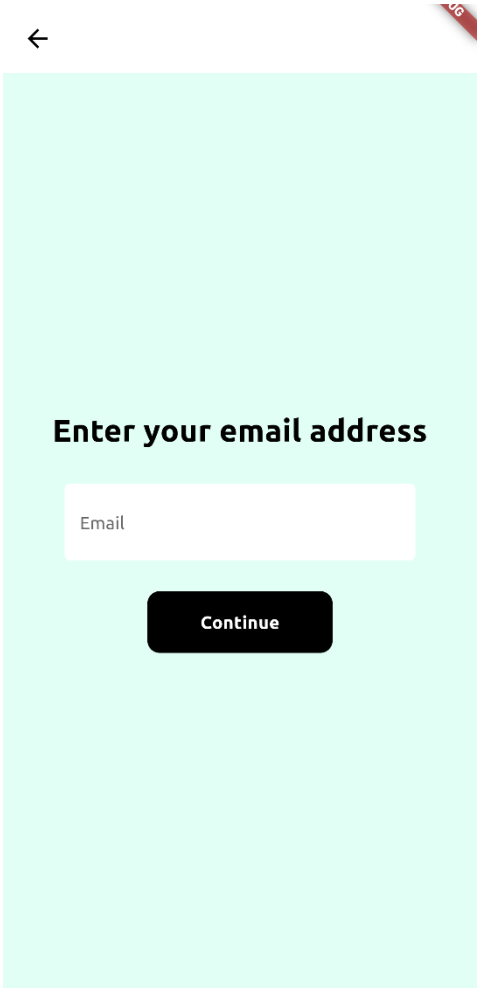
Na ovom zaslonu se također pojavljuju identične poruke kao na slici 4, te o nevažećoj email adresi i prekratkoj lozinki kao u slici 5. Osim navedenih poruka, na ovom zaslonu pojavljuju se i još neke nove poruke.



Slika 8. Poruke o greškama prilikom prijave

Ako je korisnik unio netočnu lozinku, a ispravnu email adresu, tada ga se obavještava da je unio netočnu lozinku te se moli da unese ispravnu. U drugom slučaju, ako je korisnik unio email adresu i lozinku, a ne postoji račun povezan uz tu email adresu, korisnika se također obavještava o tome. U zadnjem slučaju, kada je korisnik unio sve podatke, ispisana je poruka kako je korisnički račun onemogućen. To se može jedino dogoditi ako je administrator ove aplikacije koji ima pristup Firestore bazi podataka koja je dio Googleovog Firebasea korisnika onemogućio. Više detalja oko registracije i prijave će biti u poglavlju Autentifikacija.

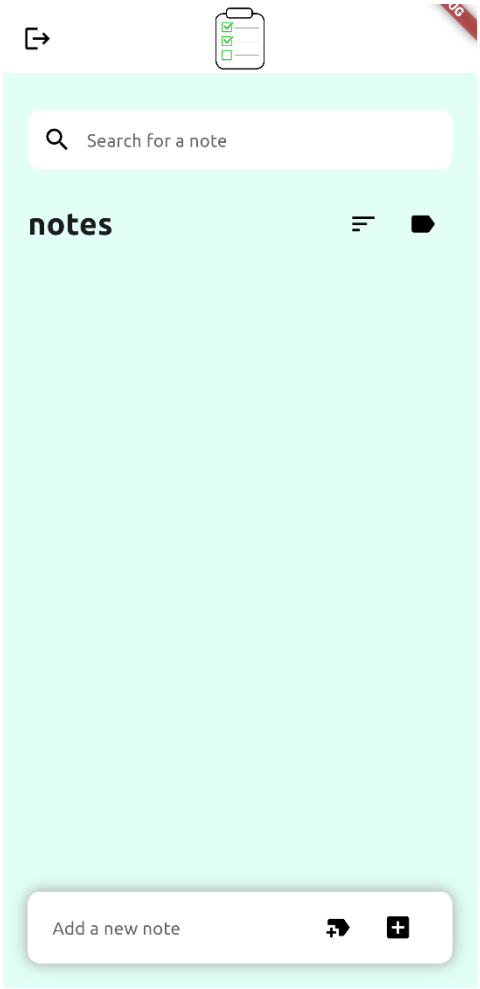
3.5. Zaslon za resetiranje lozinke



Slika 9. Korisničko sučelje zaslona za resetiranje lozinke

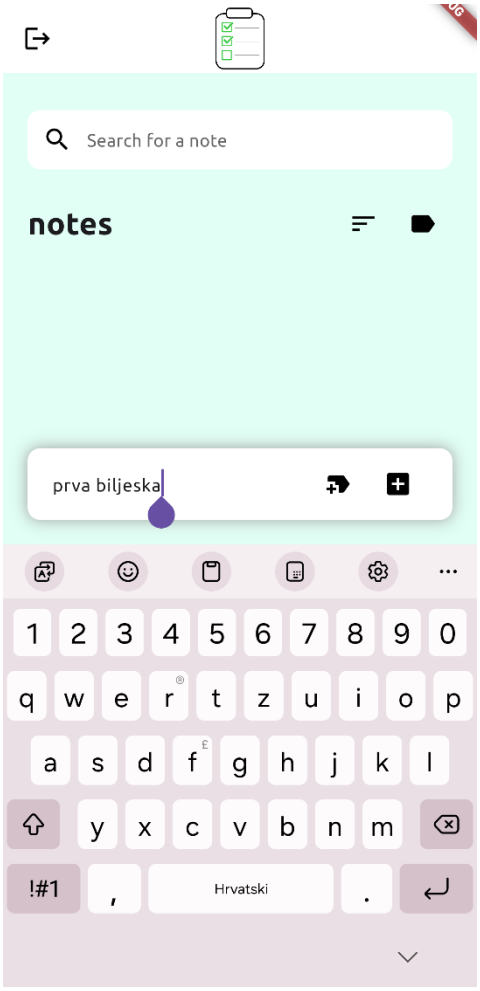
Ako je korisnik zaboravio lozinku i kliknuo na tekst „Forgot password?“ na zaslonu za prijavu, otvara se zaslon za resetiranje lozinke (Slika 9). Na zaslonu se nalazi tekst koji od korisnika očekuje unos email adrese, te ispod toga i samo polje za unos email adrese. Nakon tih elemenata nalazi se tipka „Continue“, koja potom, u slučaju ispravnog unosa, na unesenu email adresu šalje poruku s poveznicom na kojoj korisnik može promijeniti svoju lozinku. I u ovom polju vrijede pravila kao i na stranicama za registraciju i prijavu, koja su prikazana na nekima od prethodnih slika, vezana uz unos emaila te postojanje korisničkog računa s unesenom email adresom.

3.6. Glavni zaslon aplikacije



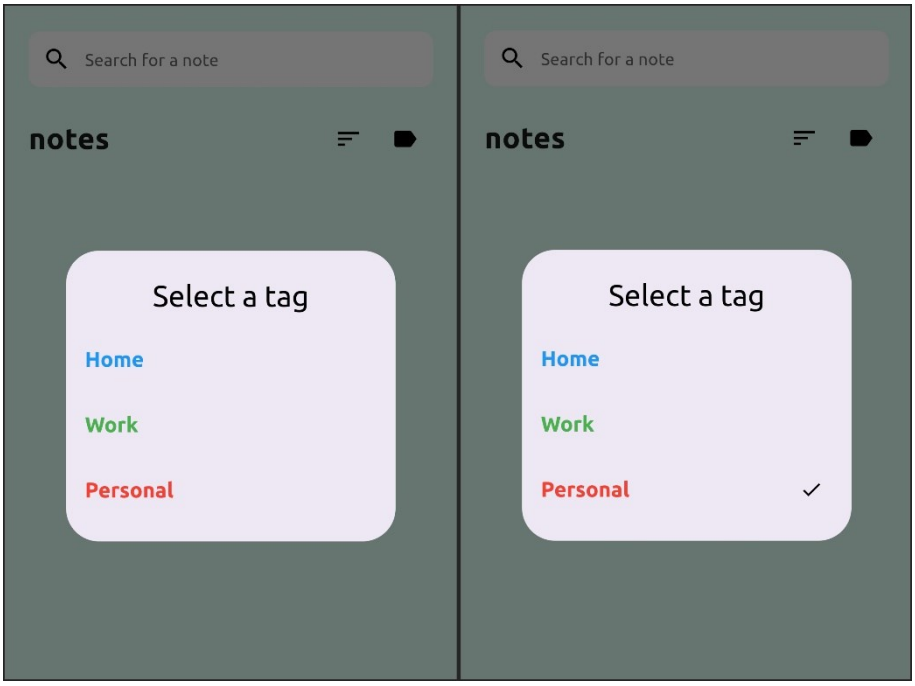
Slika 10. Korisničko sučelje glavnog zaslona aplikacije

Glavni zaslon ove aplikacije je zaslon na kojemu se vrše sve operacije vezane uz bilješke, a to su: unos teksta i odabir oznake, dodavanje bilješke, označavanje bilješke kao dovršene, brisanje bilješke, pretraživanje, sortiranje te filtriranje prema oznaci. Na vrhu zaslona nalazi se alatna traka bijele boja koja s lijeve strane ima tipku za odjavu, a na sredini se nalazi logotip aplikacije. Ispod toga se nalazi polje za unos koje služi za pretraživanje bilješki prema tekstu. Zatim ispod toga se nalazi red s naslovom „notes“ na lijevoj strani te tipkama za sortiranje i filtriranje na desnoj strani. Sredina zaslona je namijenjena za prikaz korisnikovih bilješki. Kada se korisnik tek registrirao i prvi put ušao u aplikaciju, taj dio zaslona je prazan, odnosno popunjava se dodavanjem bilješki. Ako je korisnik dodao velik broj bilješki, tada se taj dio ekrana može listati (engl. *scroll*) gore-dolje. O tome će biti više riječi u poglavlju ListView. Na donjem dijelu ekrana nalazi se polje za unos teksta bilješke, tipka za odabir oznake te bilješke i tipka za dodavanje bilješke.



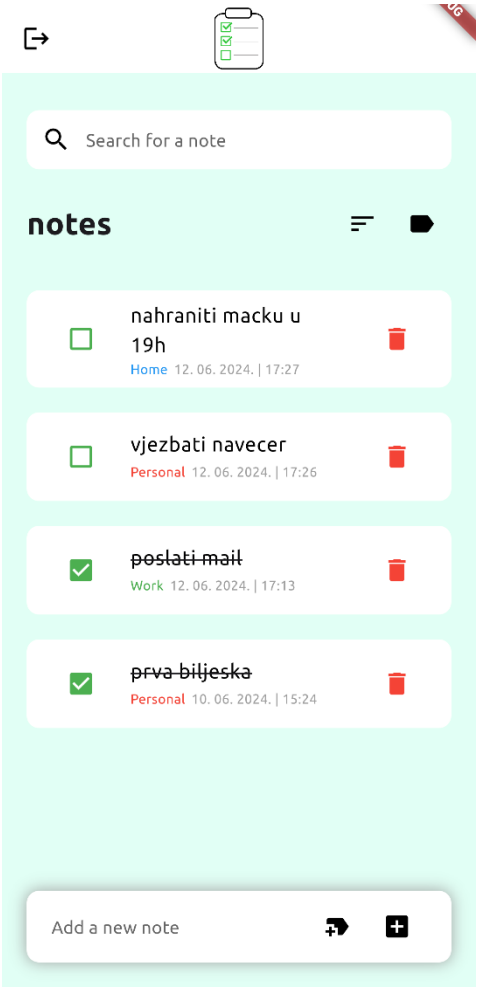
Slika 11. Primjer unosa teksta bilješke

Prilikom unosa bilješki, donji dio ekrana u kojemu se nalaze funkcionalnosti za dodavanje bilješke se podiže prilikom otvaranja tipkovnice, dok svi ostali elementi sučelja ostaju jednaki.



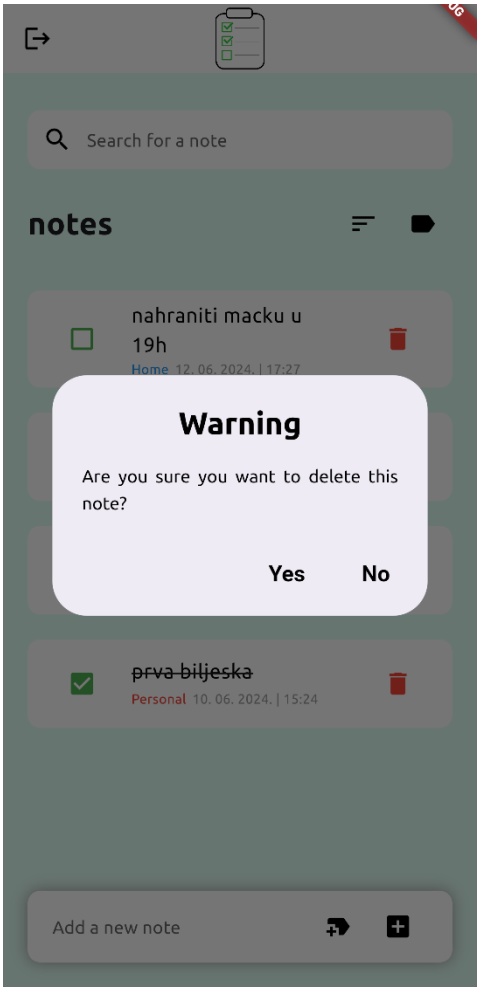
Slika 12. Prozor za odabir oznake bilješke

Pritiskom na tipku za odabir oznake, otvara se prozor koji nudi tri oznake koje se mogu dodati billješci: „Home“, „Work“ i “Personal“. Svaka od tih oznaka je označena drugom bojom kako bi se lakše raspoznavale. Odabirom oznake, ovaj prozor se zatvara. Ako se prije dodavanja bilješke ponovno pritisne tipka za odabir oznake, tada se pored odabrane oznake nalazi znak kvačice.



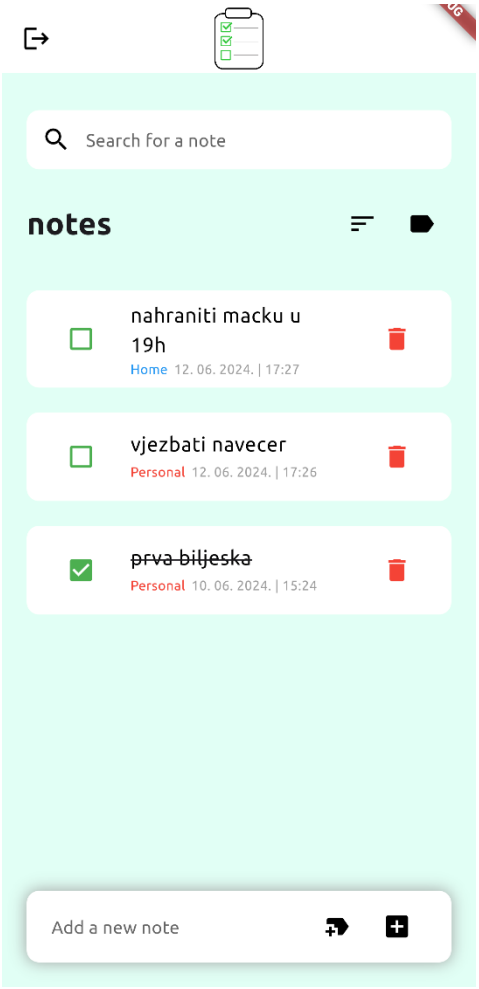
Slika 13. Korisničko sučelje glavnog zaslona s bilješkama

Bilješke se u korisničko sučelje dodaju kao retci koji sadrže tipke za upravljanje bilješkom, sadržaj bilješke te informacije o bilješci kao što su oznaka, datum dodavanja i vrijeme dodavanja. Na lijevoj strani retka nalazi se tipka s ikonom okvira za potvrdu (engl. *checkbox*) koja ima ikonu praznog okvira prilikom dodavanja bilješke. Pritiskom na tu tipku, ikona se mijenja u ikonu s kvačicom te se tekst bilješke precrtava, što označava bilješku dovršenom. Sredina retka sadržava tekst bilješke, te ispod toga oznaku koja je dodijeljena bilješci, kao i datum i vrijeme dodavanja bilješke. Na desnoj strani nalazi se tipka s ikonom smeća, koja služi za brisanje bilješke.



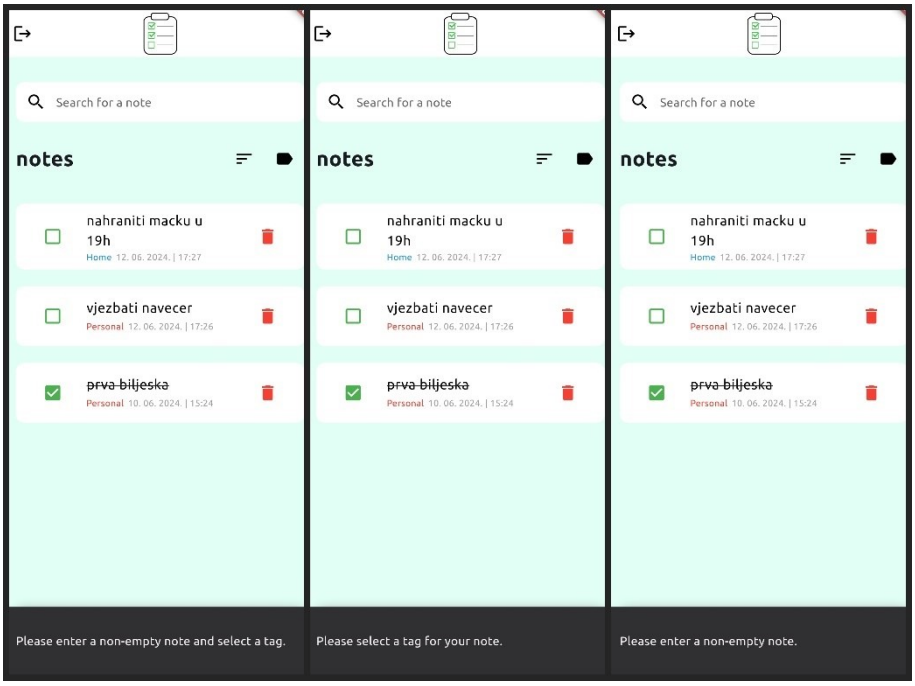
Slika 14. Prozor za brisanje bilješke

Ako korisnik stisne na ikonu smeća u želji da izbriše bilješku, prikazuje se prozor koji provjerava želi li korisnik zaista izbrisati bilješku. Korisnik može stisnuti „Yes“ ili „No“. Ako korisnik stisne „No“, prozor se zatvara i bilješka se neće izbrisati, već ostaje prikazana u korisničkom sučelju. S druge strane, ukoliko korisnik stisne „Yes“, prozor će se zatvoriti, bilješka će se izbrisati i više neće biti prikazana u korisničkom sučelju.



Slika 15. Glavni zaslon aplikacije nakon brisanja bilješke

Kada je bilješka izbrisana, poredak bilješki ostaje jednak kao i prije, samo se izbrisana bilješka maknula iz prikaza i iz Firestore baze podataka.



Slika 16. Prikaz poruka o greškama prilikom unosa bilješki

Kao što se provjeravao unos na zaslonima za registraciju i prijavu, tako se provjerava i unos i odabir oznake prilikom pritiska na tipku za dodavanje bilješki. Ako nije unesen nikakav tekst te nije odabrana oznaka za bilješku, na dnu ekrana se ispisuje poruka o tome. Isto se ponavlja i za pojedinačne slučajeve, odnosno kada samo jedna od navedenih radnji prilikom unosa nije napravljena.

Slika 17. Prozor za odjavu iz aplikacije

Ako se korisnik želi odjaviti sa svog korisničkog računa, to može učiniti pritiskom na tipku s ikonom vrata i strelice lijevo na alatnoj traci u gornjem dijelu ekrana. Tada se prikazuje prozor u kojemu se provjerava želi li se korisnik zaista odjaviti sa korisničkog računa koji ima email adresu navedenu u ovom prozoru. Ako korisnik stisne „No“, prozor se zatvara i korisnik ostaje na glavnom zaslonu aplikacije, a ako stisne „Yes“, korisnik će biti odjavljen i prebačen na početni zaslon aplikacije.

4. Implementacija

U ovome poglavlju biti će objašnjena izrada svih prethodno navedenih zaslona za ovu aplikaciju, uz isječke korištenog programskog koda.

4.1. Zasloni

4.1.1 Početni zaslon

Početni zaslon je izrađen korištenjem Stateless *widgeta*.

```
return PopScope(
  canPop: true,
  onPopInvoked: (didPop) {
    if (didPop) {
      SystemNavigator.pop();
    }
  …
);
```

U metodi za izgradnju widgeta (engl. *build*) vraća se *widget* `PopScope`, čija je svrha upravljanje pokretima unatrag pomoću tipke unatrag na sistemskoj navigaciji ili pokretom unatrag (ako korisnik na uređaju koristi navigaciju pokretima). Parametar `canPop` je postavljen na vrijednost `true`, što znači da je dopušteno vraćanje unatrag, no u `onPopInvoked` parametru je izrađena funkcija koja znači da će se pri vraćanju unatrag s ovog zaslona aplikacija ugasiti. Zadana vrijednost bez `onPopInvoked` parametra je ta da se pomoću tipke/pokreta unatrag korisnika vrati na prethodni zaslon, što bi mogao, primjerice, biti zaslon za prijavu, a to bi dovelo do nepredviđenog ponašanja aplikacije. Ovaj dio koda se još koristi i kod zaslona o uspješnoj registraciji te kod glavnog zaslona aplikacije.

```
Scaffold(
    backgroundColor: const Color.fromARGB(255, 225, 255, 245),
    body: SafeArea(
      child: Center(
        child: SingleChildScrollView(
          child: Column(
            children: […])))))
```

Kao „dijete“ `PopScope` *widgeta* postavljen je `Scaffold` *widget*, koji implementira osnovnu strukturu vizualnog izgleda materijalnog dizajna. Boja pozadine je postavljena na ranije navedenu boju čija je heksadekadska vrijednost #E1FFF5. U tijelo tog *widgeta* postavljen je *widget* `SafeArea` koji svoje „dijete“ umeće s dovoljno odmaka (engl. *padding*) kako bi se izbjeglo preklapanje s operacijskim sustavom. U ovom slučaju, s alatnom trakom sustava. Podređeni *widgeti* `Center`, `SingleChildScrollView` i `Column` su *widgeti* vezani uz sprječavanje prelijevanja (engl. *overflow*) u trenutku kada se otvara tipkovnica te namještanje pozicije na koju će se umetati sljedeći *widgeti*. Ovaj način izrade zaslona koriste svi ostali zasloni aplikacije, osim glavnog zaslona.

```
Image.asset(
  'lib/images/you_to_do.png',
  width: 125,
  height: 125,
)
```

Konstruktor `Image.asset` stvara *widget* koji prikazuje sliku koja je definirana među resursima (engl. *assets*). Prikazuje se logotip aplikacije, visine i širine 125 piksela.

```
const SizedBox(height: 25)
```

Za potrebe urednijeg i preglednijeg rasporeda elemenata, tijekom svih zaslona često je korišten *widget* `SizedBox` koji stvara prazan okvir sa određenom visinom ili širinom, u pikselima.

```
Text('Welcome to you to-do!',
    style: GoogleFonts.ubuntu(
        color: Colors.black,
        fontSize: 28,
        fontWeight: FontWeight.bold))
```

Widget `Text` koristi se za ispis određenog teksta uz mogućnosti uređivanja stila. Svi tekstovi u aplikaciji su pisani Ubuntu fontom. Naslovni tekst dobrodošlice na početnom zaslonu je crne boje, podebljan je te je veličine 28. Jednaki stil imaju i naslovni tekstovi na svim ostalim zaslonima.

```
Text('Start writing the notes to your to-do list today!',
    style: GoogleFonts.ubuntu(
        color: Colors.grey, fontSize: 16))
```

Manja poruka ima font sive boje i veličine 16.

```
const Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Padding(
        padding: EdgeInsets.symmetric(
            horizontal: 15, vertical: 25)),
    Expanded(child: SignupHomeButton()),
    Padding(
        padding: EdgeInsets.symmetric(
            horizontal: 5, vertical: 25)),
    Expanded(child: LoginHomeButton()),
    Padding(
        padding: EdgeInsets.symmetric(
            horizontal: 15, vertical: 25))
  ])
```

Kao zadnji *widget* na ovom zaslonu koristi se *widget* `Row` koji je postavljen na sredinu zaslona ispod poruka, a služi za postavljanje tipki za registraciju i prijavu u isti red. Tipke su jednake veličine zbog korištenja `Expanded` *widgeta*, te postoji jednak razmak s lijeve i desne strane od tipki, što je postignuto korištenjem `Padding` *widgeta*. Između tipki je također razmak.

4.1.2 Zaslon za registraciju

Za izradu je korišten Stateful *widget*. Izrađen je *state* objekt koji sadrži sljedeći dio koda.

```
final emailController = TextEditingController();
final passwordController = TextEditingController();
var obscuredPassword = true;

@override
void dispose() {
  emailController.dispose();
  passwordController.dispose();
  super.dispose();
}
```

Na početku su definirani kontroleri za unos emaila i lozinke, te varijabla koja će biti korištena za upravljanje fontom lozinke, odnosno je li šifrirana ili ne, prilikom pritiska tipke u polju za unos. Zatim je korištena metoda `dispose` u kojoj se odjavljuju oba kontrolera.

```
return Scaffold(
  appBar: AppBar(
    backgroundColor: Colors.white,
    leading: IconButton(
      icon: const Icon(Icons.arrow_back, color: Colors.black),
      onPressed: () {
        Navigator.push(context,
            MaterialPageRoute(builder: (context) => const FirstPage()));
      }))
```

Kod izgradnje *widgeta* se vraća `Scaffold` *widget* koji sadrži `AppBar`, odnosno alatnu traku s tipkom za povratak na prethodni zaslon u hijerarhiji aplikacije, što je u ovom slučaju povratak na početni zaslon. Ovaj princip se također koristi i kod zaslona za prijavu te kod zaslona za resetiranje lozinke.

```
GestureDetector(
  onTap: () async {
    await signUp(context, emailController, passwordController);
  },
  child: Container(
    width: 150,
    height: 50,
    decoration: BoxDecoration(
        color: Colors.black,
        borderRadius: BorderRadius.circular(10)),
    child: Center(
      child: Text("Sign up",
          style: GoogleFonts.ubuntu(
              color: Colors.white,
              fontSize: 16,
              fontWeight: FontWeight.bold)),
    )))
```

Nakon polja za unos email adrese i lozinke (koji će biti objašnjeni u poglavlju Unos i forme), ključan element ovog zaslona je tipka za registraciju. Tipka je napravljena pomoću *widgeta* `GestureDetector` koji detektira kada korisnik pritisne ovu tipku i tada vrši funkciju `signUp` koja pokušava registrirati korisnika (više riječi o toj funkciji u poglavlju Autentifikacija). Izgled tipke je kreiran pomoću `Container` *widgeta* s dekoracijom koja postavlja boju tipke na crnu te zaobljuje rubove. Tekst „Sign up“ je bijele boje i fonta 16 te je podebljan.

4.1.3 Zaslon o uspješnoj registraciji

Zaslon o uspješnoj registraciji je izrađen pomoću Stateless *widgeta*. Pritiskom tipke za unatrag ili pokretom unatrag, aplikacija se ne vraća na prethodni zaslon, već se izlazi iz aplikacije, na način kako je i ranije već prikazano.

```
const Icon(Icons.check_circle,
    color: Colors.green, size: 100),

const SizedBox(height: 15),

Padding(
```

```
  padding: const EdgeInsets.symmetric(horizontal: 25),
  child: Text(
    'Thanks for signing up to you to-do!',
    style: GoogleFonts.ubuntu(
        color: Colors.black,
        fontSize: 28,
        fontWeight: FontWeight.bold),
    textAlign: TextAlign.center),
)
```

Element koji je najuočljiviji na ovom zaslonu je ikona kvačice. Ikona je kreirana korištenjem `Icon` *widgeta*, zelene je boje i veličine 100 piksela. Nakon ikone je korišten `SizedBox` *widget* za prazan prostor između ikone i teksta koji slijedi. S obzirom da je naslovni tekst na ovom zaslonu prilično dugačak, umotan je u *widget* `Padding`, koji dodaje horizontalni odmak veličine 25 piksela, čime se dio teksta gura u novi red.

```
GestureDetector(
    onTap: () {
      Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => const HomePage()));
    },
    child: Container(
      width: 150,
      height: 50,
      decoration: BoxDecoration(
          color: Colors.black,
          borderRadius: BorderRadius.circular(10)),
      child: Center(
        child: Text("OK",
            style: GoogleFonts.ubuntu(
                color: Colors.white,
                fontSize: 16,
                fontWeight: FontWeight.bold)),
      )))
```

Na kraju se nalazi tipka „OK“ koja je napravljena uz *widget* `GestureDetector`, te kada korisnik pritisne ovu tipku prebacuje ga se na glavni zaslon aplikacije.

4.1.4 Zaslon za prijavu

Mnogi elementi ovog zaslona jednaki su kao i na ranije navedenim zaslonima, kao što su alatna traka, stiliziranje teksta, polja za unos email adrese i lozinke.

```
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 50),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
      GestureDetector(
        onTap: () {
          Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => const EmailInputPage()));
        },
        child: Text('Forgot password?',
            style: GoogleFonts.ubuntu(
                color: const Color.fromARGB(255, 90, 90, 90),
```

```
                fontWeight: FontWeight.bold)),
      )]))
```

Tekst „Forgot password?“ pozicioniran je ispod polja za unos lozinke. Korišten je *widget* `Padding` kako bi se tekst odmaknuo od lijevog ruba ekrana. Tekst je tamno sive boje, a vodi na zaslon za resetiranje lozinke.

```
GestureDetector(
  onTap: () async {
    await logIn(context, emailController, passwordController);
  },
  child: Container(
    width: 150,
    height: 50,
    decoration: BoxDecoration(
        color: Colors.black,
        borderRadius: BorderRadius.circular(10)),
    child: Center(
      child: Text("Log in",
          style: GoogleFonts.ubuntu(
              color: Colors.white,
              fontSize: 16,
              fontWeight: FontWeight.bold)),
    )))
```

Kao i ranije navedene tipke, i tipka „Log in“ koristi `GestureDetector` *widget*, te pod parametrom `onTap` se poziva funkcija `logIn` koja pokušava prijaviti korisnika u aplikaciju. Detaljnije će navedena funkcija biti opisana u poglavlju Autentifikacija.

4.1.5 Zaslon za resetiranje lozinke

Ovaj zaslon također kao i zaslon za registraciju i prijavu koristi Stateful *widget*, s obzirom da se i na ovom zaslonu vrši unos email adrese.

```
GestureDetector(
  onTap: () async {
    await passwordReset(context, emailController);
  },
  child: Container(
    width: 150,
    height: 50,
    decoration: BoxDecoration(
        color: Colors.black,
        borderRadius: BorderRadius.circular(10)),
    child: Center(
      child: Text("Continue",
          style: GoogleFonts.ubuntu(
              color: Colors.white,
              fontSize: 16,
              fontWeight: FontWeight.bold)),
    )))
```

U programskom kodu ovog zaslona ponavljaju se mnogi elementi koji su već ranije navedeni. Tako se ponavlja i `GestureDetector` *widget* koji služi za kreiranje tipke s tekstom „Continue“. Razlika kod ovog `GestureDetector` je što poziva funkciju `passwordReset`, koja na unesenu email adresu (ako je ispravna) šalje poruku u kojoj korisnik dobiva poveznicu na kojoj može promijeniti svoju lozinku. Dekoracija izgleda tipke te tekst i stil istog su jednaki kao i na tipkama koje se nalaze na prethodnim zaslonima.

4.1.6 Glavni zaslon aplikacije

```
IconButton(
  icon: const Icon(Icons.logout, color: Colors.black),
  onPressed: () async {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) {
        return PopScope(
          canPop: false,
          child: AlertDialog(
            title: const Text('Warning', textAlign: TextAlign.center),
            titleTextStyle: GoogleFonts.ubuntu(
                color: Colors.black, fontWeight: FontWeight.bold,
                fontSize: 28),
            content: Text(
                'Are you sure you want to log out from the account with the
email address: $email?',
                textAlign: TextAlign.justify),
            contentTextStyle: GoogleFonts.ubuntu(
                color: Colors.black, fontSize: 16, height: 1.5),
            backgroundColor: Colors.white,
            actionsPadding:
                const EdgeInsets.only(bottom: 10, right: 10),
            actions: [
              TextButton(
                child: const Text('Yes',
                    style: TextStyle(
                        color: Colors.black,
                        fontWeight: FontWeight.bold,
                        fontSize: 20)),
                onPressed: () async {
                  await _auth.signOut();
                  if (mounted) {
                    Navigator.pushAndRemoveUntil(
                        contextNew,
                        MaterialPageRoute(
                            builder: (context) => const FirstPage()),
                        (route) => false);
                  }
                },
              ),
              TextButton(
                child: const Text('No',
                    style: TextStyle(
                        color: Colors.black,
                        fontWeight: FontWeight.bold,
                        fontSize: 20)),
                onPressed: () {Navigator.of(context).pop();},
              )
            ],
          ),
        );
      });})
```

Na vrhu glavnog zaslona aplikacije nalazi se alatna traka napravljena korištenjem *widgeta* `AppBar`, koja je bijele boje i u sredini sadrži logotip aplikacije, a na lijevoj strani se nalazi tipka za odjavu iz aplikacije. Tipka je kreirana pomoću *widgeta* `IconButton` s prikladnom ikonom. Pritiskom na tipku otvara se prozor koji upozorava korisnika i provjerava želi li se

zaista odjaviti iz aplikacije, sa korisničkog računa čija je email adresa također navedena u ovoj poruci. Ako korisnik stisne na „No“, prozor se zatvara te se korisnik ne odjavljuje i ostaje na glavnom zaslonu. Ako stisne na „Yes“, poziva se funkcija `signOut`, koja se nalazi u Firebase paketu, te se korisnik odjavljuje iz aplikacije i vraća ga se na početni zaslon. Ispod alatne trake nalazi se polje za pretraživanje, o kojemu će više riječi biti u poglavlju ListView.

Ispod polja za pretraživanje, nalazi se red kreiran pomoću *widgeta* `Row` u kojemu se nalazi tekst „notes“ koji označava početak dijela sučelja u kojemu će se nalaziti dodane bilješke. Unutar tog istog reda kreiran je još jedan red korištenjem *widgeta* `Row` u kojemu se nalaze tipke za sortiranje i filtriranje bilješki. Obje funkcionalnosti će opširnije biti objašnjene u poglavlju ListView.

```
return Container(
  margin: const EdgeInsets.only(
      bottom: 20, left: 20, right: 20),
  padding: const EdgeInsets.symmetric(horizontal: 20),
  decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(10)),
  …
);
```

Vizualni prikaz bilješki u sučelju se stvara korištenjem *widgeta* `Container`. Bijele je boje, zaobljenih rubova te sadrži nekoliko modifikacija oko margina i odmaka kako ne bi dodirivao rubove, te kako bi se između svake bilješke postojao razmak.

```
Row(
  children: [
    IconButton(
        icon: Icon(
          note['isDone']
              ? Icons.check_box
              : Icons.check_box_outline_blank,
          color: Colors.green,
        ),
        onPressed: () async {
          bool isDone = !note['isDone'];
          await DatabaseService(uid: uid)
              .updateNoteIsDone(note.id, isDone);
        }),
    …
  ],
)
```

Kao „dijete“, *widget* `Row` služi da popuni *widget* `Container` sa sadržajem, odnosno s nekolicinom ostalih *widgeta* kako bi se napravio prikaz sadržaja unesenih bilješki te tipke koje služe za upravljanje bilješkom. Prvi widget koji se koristi je widget `IconButton` za kreiranje tipke s ikonom okvira za potvrdu koja služi za označavanje bilješke kao dovršene. Kada korisnik stisne na ovu tipku, ikona se mijenja u ikonu s kvačicom, prekriži se tekst bilješke te se varijabli `isDone` vrijednost postavlja na `true` u Firestore bazi podataka.

```
Expanded(
  child: ListTile(
    title: Text(
      '${note['text']}',
      style: GoogleFonts.ubuntu(
```

```
          fontSize: 18,
          color: Colors.black,
          decoration: note['isDone']
              ? TextDecoration.lineThrough
              : null,
        ),
      ),
      subtitle: Row(
        children: [
          Text(
            '${note['tag']}',
            style: GoogleFonts.ubuntu(
              fontSize: 12,
              color: findTagColor(
                  note['tag'], availableTags),
            ),
          ),
          const SizedBox(width: 5),
          Text(
            '${note['formattedDateTime']}',
            style: GoogleFonts.ubuntu(
              fontSize: 12,
              color: Colors.grey,
            ),
          ),
        ],
      ),
    ),
  ),
),
```

Za prikaz teksta bilješki, korišten je *widget* `Expanded` koji kao „dijete“ ima *widget* `ListTile`. Kao naslov `ListTile` *widgeta* postavlja se tekst bilješke, a kao podnaslov, odnosno tekst ispod teksta bilješke, postavljene su informacije o oznaci koja je dodijeljena bilješki te datum i vrijeme dodavanja bilješke. Tekst bilješke je crne boje i većeg fonta, tekst oznake bilješke je boje oznake koja joj je dodijeljena (crvena, plava ili zelena, ovisno o oznaci), dok je tekst sa datumom i vremenom dodavanja bilješke sive boje.

```
IconButton(
  color: Colors.red,
  icon: const Icon(Icons.delete),
  onPressed: () async {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) {
        return PopScope(
          canPop: false,
          child: AlertDialog(
            title: const Text('Warning',
                textAlign: TextAlign.center),
            titleTextStyle: GoogleFonts.ubuntu(
                color: Colors.black,
                fontWeight: FontWeight.bold,
                fontSize: 28),
            content: const Text(
                'Are you sure you want to delete this note?',
                textAlign: TextAlign.justify),
            contentTextStyle: GoogleFonts.ubuntu(
                color: Colors.black,
```

```
                    fontSize: 16,
                    height: 1.5),
                backgroundColor: Colors.white,
                actionsPadding: const EdgeInsets.only(
                    bottom: 10, right: 10),
                actions: [
                  TextButton(
                    child: const Text('Yes',
                        style: TextStyle(
                            color: Colors.black,
                            fontWeight: FontWeight.bold,
                            fontSize: 20)),
                    onPressed: () async {
                      Navigator.of(context).pop();
                      await DatabaseService(uid: uid)
                          .deleteNote(note.id);
                    },
                  ),
                  TextButton(
                    child: const Text('No',
                        style: TextStyle(
                            color: Colors.black,
                            fontWeight: FontWeight.bold,
                            fontSize: 20)),
                    onPressed: () {
                      Navigator.of(context).pop();
                    },
                  )
                ],
              ),
            );
          },
        );
      },
)
```

Posljednji element kod prikaza bilješki je tipka za brisanje bilješke, također napravljena pomoću *widgeta* `IconButton`. Tipka ima ikonu smeća crvene boje, a pritiskom na tipku otvara se prozor za brisanje bilješke (slika 14), napravljen korištenjem *widgeta* `AlertDialog`. Ako se stisne tipka „Yes“, bilješka se briše iz sučelja, kao i iz Firestore baze podataka, a ako se stisne tipka „No“, prozor se zatvara, bilješka se ne briše i korisnik ostaje na glavnom zaslonu aplikacije.

Polje za pretraživanje, tipka za sortiranje, tipka za filtriranje te polje za dodavanje bilješki biti će opisani u zasebnim potpoglavljima.

4.2. Autentifikacija

U ovom poglavlju biti će detaljnije opisani elementi sučelja vezani uz autentifikaciju, kao i sama implementacija autentifikacije.

4.2.1 Registracija

Prilikom registracije, poziva se funkcija `signUp`. Funkcija sadrži `try` metodu u kojoj se nalaze funkcionalnosti vezane uz registraciju korisničkog računa, a ako dođe do neke pogreške, poziva se programski kod iz `on` metode u kojemu su definirane sve moguće pogreške koje

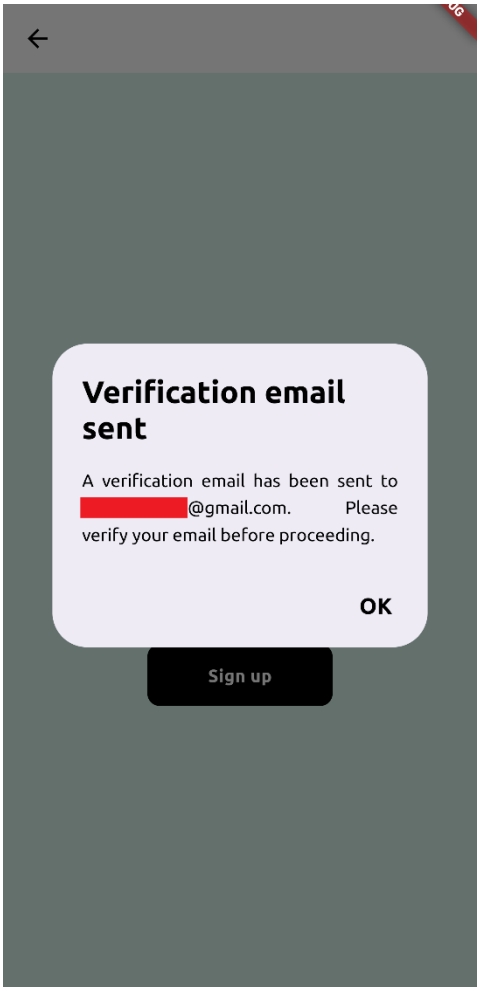
Google Firebase nudi prilikom registracije, te se ispisuju u prozoru kao što se može vidjeti na slici 4 i slici 5.

```
Future<void> signUp(BuildContext context, TextEditingController
emailController, TextEditingController passwordController) async {
  String email = emailController.text.trim();
  String password = passwordController.text.trim();

  try {
    UserCredential credential = await FirebaseAuth.instance
        .createUserWithEmailAndPassword(email: email, password: password);

    await credential.user!.sendEmailVerification();
```

Funkcija kao argumente uzima trenutni kontekst, te kontrolere za unos email adrese i lozinke. Uneseni tekst email adrese i lozinke se krati funkcijom `trim` kako bi se spriječio slučajan ili namjeran unos razmaka na kraju. Nakon toga se ulazi u `try` metodu u kojoj se prvo poziva funkcija `createUserWithEmailAndPassword` za unesenu email adresu i lozinku. U slučaju netočnog unosa, tada se ispisuju pogreške ranije navedene u `on` metodi. U protivnom, korisniku se na unesenu email adresu šalje mail za verifikaciju, što je u kodu napravljeno pozivom funkcije `sendEmailVerification`.

```
showDialog(
  context: context,
  barrierDismissible: false,
  builder: (BuildContext context) {
    return PopScope(
      canPop: false,
      child: AlertDialog(
        title: const Text('Verification email sent'),
        titleTextStyle: GoogleFonts.ubuntu(
            color: Colors.black,
            fontSize: 28,
            fontWeight: FontWeight.bold),
        content: Text(
            'A verification email has been sent to $email. Please verify
your email before proceeding.',
            textAlign: TextAlign.justify),
        contentTextStyle: GoogleFonts.ubuntu(
            color: Colors.black, fontSize: 16, height: 1.5),
        backgroundColor: Colors.white,
        actionsPadding: const EdgeInsets.only(bottom: 10, right: 10),

        …

      ));
  });
```

Kada je mail za verifikaciju poslan, otvara se prozor s porukom koja obavještava korisnika da je mail poslan na unesenu email adresu te ga se moli da verificira svoju email adresu prije nastavka. Ovaj prozor se ne može zatvoriti pritiskom tipke za povratak na uređaju kako bi korisnik bio primoran provjeriti svoj mail i verificirati ga pritiskom na poveznicu u poruci koja mu je poslana.



Slika 18. Prozor o poslanom mailu za verifikaciju

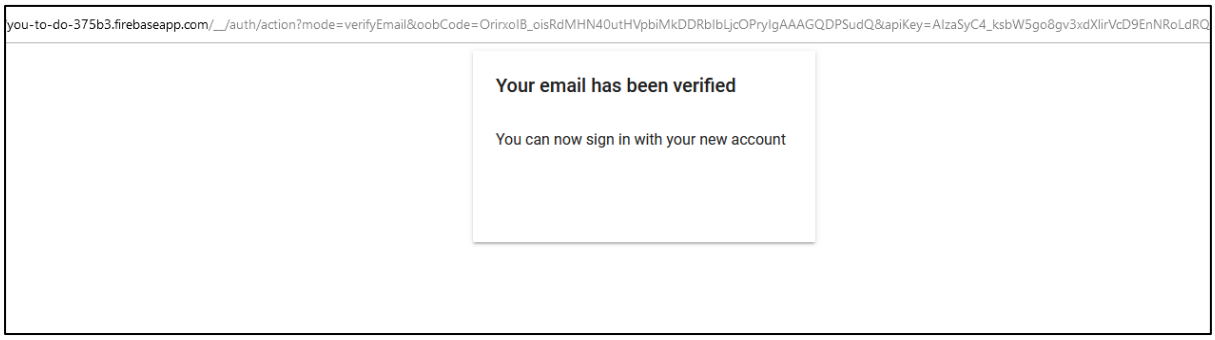
```
TextButton(
  child: Text('OK',
      style: GoogleFonts.ubuntu(
          color: Colors.black,
          fontSize: 20,
          fontWeight: FontWeight.bold)),
  onPressed: () async {
    FirebaseAuth.instance.authStateChanges().listen(
      (User? user) async {
        await credential.user!.reload();

        if (user != null && user.emailVerified) {
          await FirebaseAuth.instance
              .signInWithEmailAndPassword(
                  email: email, password: password);
          if (context.mounted) {
            Navigator.push(context,
                MaterialPageRoute(
                    builder: (context) =>
                        const SignupSuccessPage()));
          }
        }
        …
      },
    );
```

```
    },
)
```

U prozoru na slici 18 također postoji i tipka „OK“ koja je praktički jedini način da korisnik napusti ovaj zaslon (osim prisilnog zatvaranja aplikacije). Kada korisnik pritisne tipku, osvježava se stanje u Firebase-u te se provjerava ako je korisnik verificirao svoju email adresu. Ako jest, poziva se funkcija `signInWithEmailAndPassword` koja korisnika prijavljuje u aplikaciju i prebacuje ga na zaslon o uspješnoj registraciji.

Slika 19. Poruka za verifikaciju email adrese

Na slici iznad nalazi se snimka zaslona koja prikazuje kako izgleda automatizirana poruka koja se korisniku šalje kada treba verificirati svoju email adresu. U poruci se nalazi poveznica na koju korisnik mora kliknuti želi li biti verificiran.



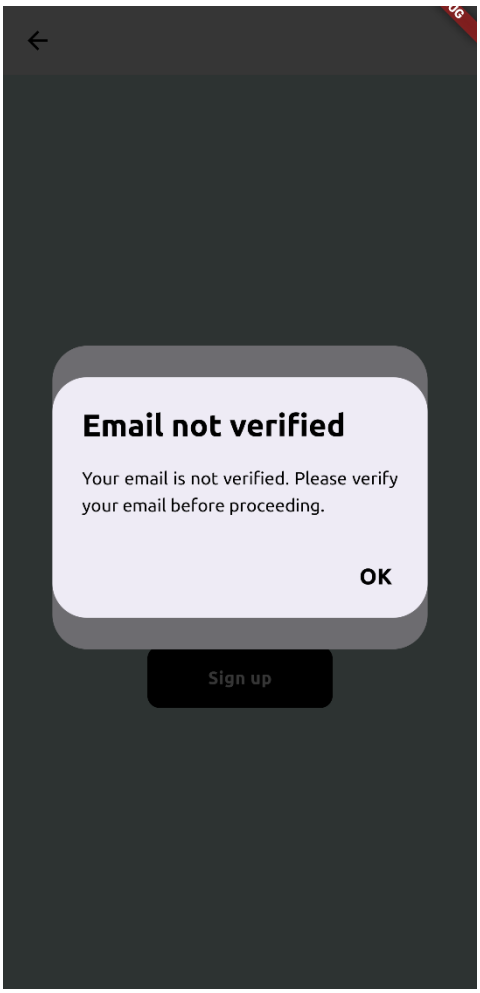
Slika 20. Uspješna verifikacija email adrese

Kada korisnik klikne na poveznicu, automatski postaje verificiran te se na poveznici ispisuje poruka kako je korisnik verificiran i kako se sada može prijaviti u aplikaciju sa svojim korisničkim računom.

```
showDialog(
  context: context,
  builder: (BuildContext context) {
    return PopScope(
      canPop: false,
      child: AlertDialog(
        title: const Text('Email not verified'),
        titleTextStyle: GoogleFonts.ubuntu(
            color: Colors.black,
            fontSize: 28,
            fontWeight: FontWeight.bold),
        content: const Text(
            'Your email is not verified. Please verify your email before
proceeding.',
            textAlign: TextAlign.justify),
```

```
          contentTextStyle: GoogleFonts.ubuntu(
              color: Colors.black, fontSize: 16, height: 1.5),
          backgroundColor: Colors.white,
          actionsPadding: const EdgeInsets.only(bottom: 10, right: 10),
          actions: [
            TextButton(
              child: Text('OK',
                  style: GoogleFonts.ubuntu(
                      color: Colors.black,
                      fontSize: 20,
                      fontWeight: FontWeight.bold)),
              onPressed: () {Navigator.pop(context);},
            )]));
    },
);
```

Ako korisnik stisne na tipku „OK“ bez verifikacije svoje email adrese, otvara se novi prozor koji obavještava kako nije verificirana email adresa te se moli da to učini.



Slika 21. Poruka o email adresi koja nije verificirana

4.2.2 Prijava

Kod prijave u aplikaciju poziva se funkcija `logIn`. I ova funkcija također sadrži `try` metodu u kojoj se nalaze funkcionalnosti vezane uz prijavu te, isto kao i kod registracije, ako dođe do neke pogreške, poziva se programski kod iz `on` metode u kojemu su definirane sve moguće

pogreške koje Google Firebase nudi prilikom prijave, koje se tada ispisuju u prozoru (slike 4, 5, 8).

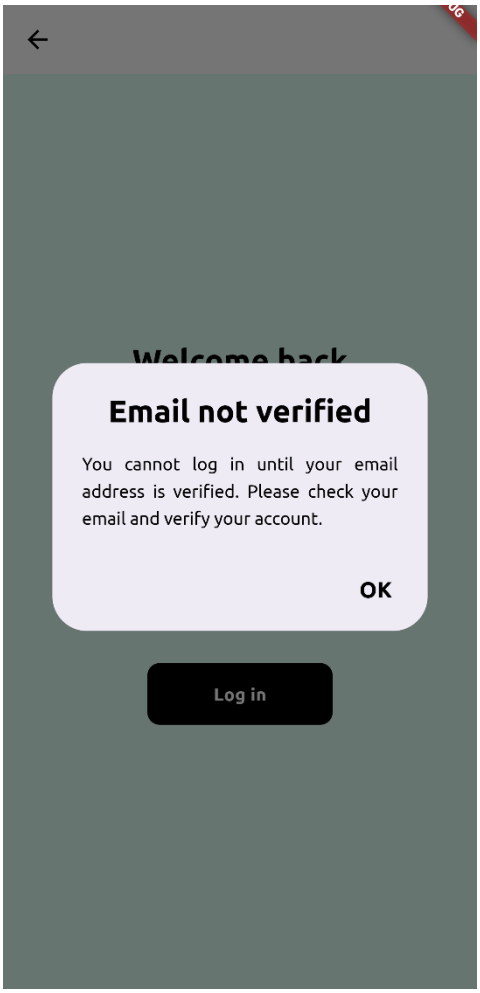
```
Future<void> logIn(BuildContext context, TextEditingController
emailController, TextEditingController passwordController) async {
  String email = emailController.text.trim();
  String password = passwordController.text.trim();

  try {
    UserCredential credential = await FirebaseAuth.instance
        .signInWithEmailAndPassword(email: email, password: password);
    User? user = credential.user;

    if (user != null) {
      await user.reload();

      if (user.emailVerified && context.mounted) {
        Navigator.push(context,
          MaterialPageRoute(builder: (context) => const HomePage()),
        );
      }
```

Kao i funkcija za registraciju, i ova funkcija kao argumente uzima trenutni kontekst, te kontrolere za unos email adrese i lozinke, koje tada krati funkcijom `trim`. Zatim se poziva `try` metoda u kojoj se na početku poziva funkcija `signInWithEmailAndPassword` za prijavu. Ako je korisnik unio svoje točne podatke te ako je verificirao svoju email adresu, tada ulazi na glavni zaslon aplikacije na kojemu može upravljati vlastitim bilješkama.

```
showDialog(
  context: context,
  barrierDismissible: false,
  builder: (BuildContext context) {
    return PopScope(
      canPop: false,
      child: AlertDialog(
        title: const Text('Email not verified',
            textAlign: TextAlign.center),
        titleTextStyle: GoogleFonts.ubuntu(
            color: Colors.black,
            fontWeight: FontWeight.bold,
            fontSize: 28),
        content: const Text(
            'You cannot log in until your email address is verified. Please
check your email and verify your account.',
            textAlign: TextAlign.justify),
        contentTextStyle: GoogleFonts.ubuntu(
            color: Colors.black, fontSize: 16, height: 1.5),
        backgroundColor: Colors.white,
        actionsPadding: const EdgeInsets.only(bottom: 10, right: 10),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK',
                style: GoogleFonts.ubuntu(
                    color: Colors.black,
                    fontSize: 20,
                    fontWeight: FontWeight.bold)),
          )
```

```
          ],
        ),
      );
    },
);
```

Ako se korisnik pokušava prijaviti iako prethodno nije verificirao svoju email adresu (što je moguće jedino ako je korisnik prilikom registracije prisilno zatvorio aplikaciju te se pri sljedećem ulasku u aplikaciju pokušao prijaviti u aplikaciju), otvara se prozor u kojemu se ispisuje poruka kako se ne može prijaviti u aplikaciju sve dok ne verificira svoju email adresu. Pritiskom na tipku „OK“ ovaj prozor se zatvara, te korisnik i dalje ostaje na zaslonu za prijavu.



Slika 22. Pokušaj prijave bez verifikacije email adrese

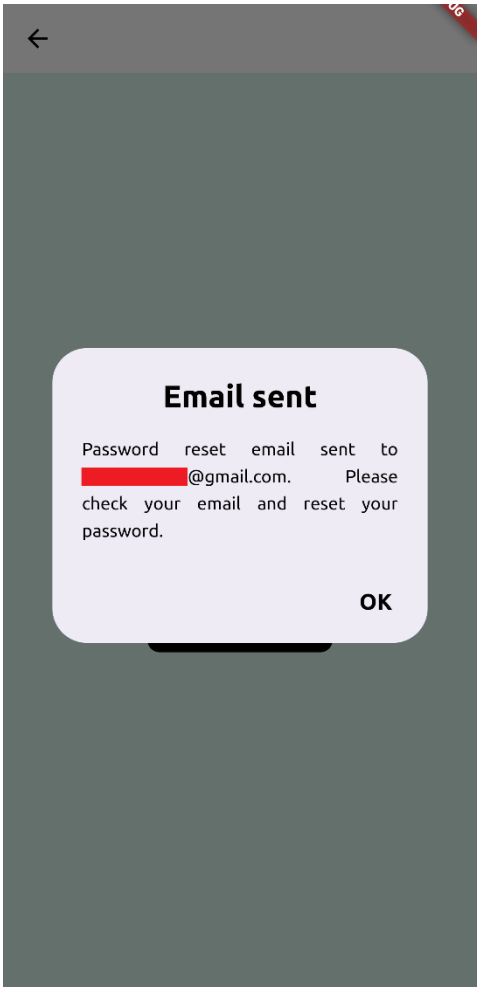
4.2.3 Resetiranje lozinke

Ako je korisnik zaboravio svoju lozinku ili ju iz nekog drugog razloga želi promijeniti, to može učiniti na zaslonu za resetiranje lozinke. Sve što je potrebno je da korisnik unese email adresu vlastitog korisničkog računa (naravno, ako ga je kreirao prije tog).

```
Future<void> passwordReset(
    BuildContext context, TextEditingController emailController) async {
  String email = emailController.text.trim();

  try {await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
```

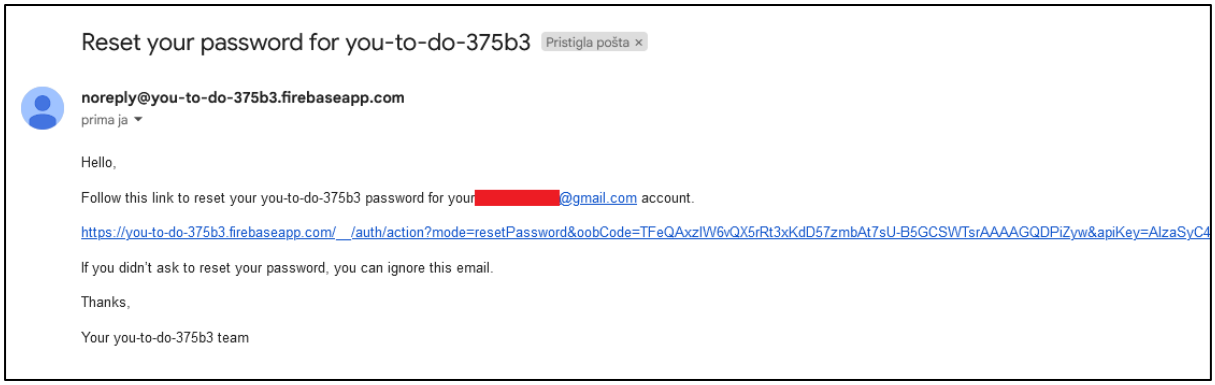
Funkcija za resetiranje lozinke nazvana je `passwordReset`. Kao argumente uzima trenutni kontekst i kontroler za unos email adrese. Nakon toga se u varijablu `email` sprema unesena email adresa skraćena korištenjem funkcije `trim`. Prvo što se radi u `try` metodi je pozivanje funkcije `sendPasswordResetEmail` koja na unesenu email adresu šalje poruku s poveznicom za resetiranje lozinke. Ako nije unesena email adresa u polje, ako je unesena netočna email adresa ili ako ne postoji korisnički račun povezan s unesenom email adresom, ispisuju se odgovarajuće poruke u novom prozoru.

```
showDialog(
  context: context,
  barrierDismissible: false,
  builder: (BuildContext context) {
    return PopScope(
      canPop: false,
      child: AlertDialog(
        title: const Text('Email sent', textAlign: TextAlign.center),
        titleTextStyle: GoogleFonts.ubuntu(
            color: Colors.black,
            fontWeight: FontWeight.bold,
            fontSize: 28),
        content: Text(
            'Password reset email sent to $email. Please check your email
and reset your password.',
            textAlign: TextAlign.justify),
        contentTextStyle: GoogleFonts.ubuntu(
            color: Colors.black, fontSize: 16, height: 1.5),
        backgroundColor: Colors.white,
        actionsPadding: const EdgeInsets.only(bottom: 10, right: 10),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK',
                style: GoogleFonts.ubuntu(
                    color: Colors.black,
                    fontSize: 20,
                    fontWeight: FontWeight.bold)),
          )
        ],
      ),
    );
  },
);
```



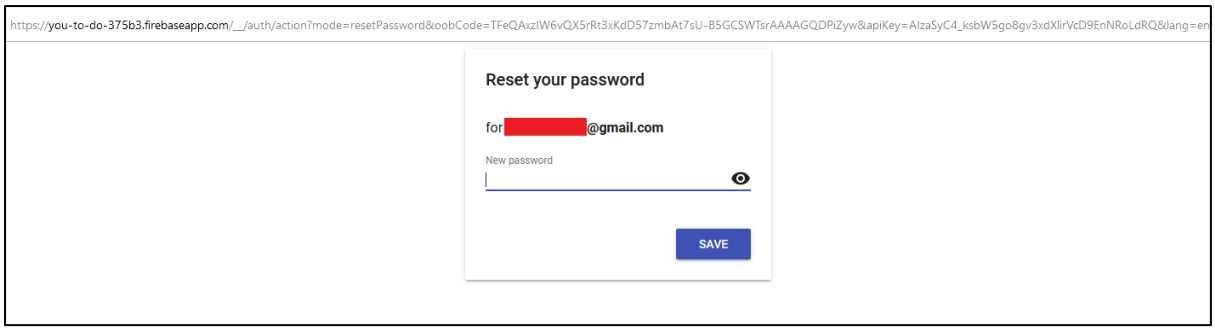
Slika 23. Prozor o poslanom mailu za resetiranje lozinke

Kada je unesena točna email adresa, ispisuje se poruka na slici iznad, u kojoj je korisnik obaviješten da mu je poslana poruka o resetiranju lozinke. Pritiskom na tipku „OK“, korisnik ostaje na zaslonu za resetiranje lozinke, ali može se vratiti na zaslon za prijavu, kao i na početni zaslon. Korisniku je omogućena prijava sa starom lozinkom sve dok ju ne odluči promijeniti, što također nije obavezno.



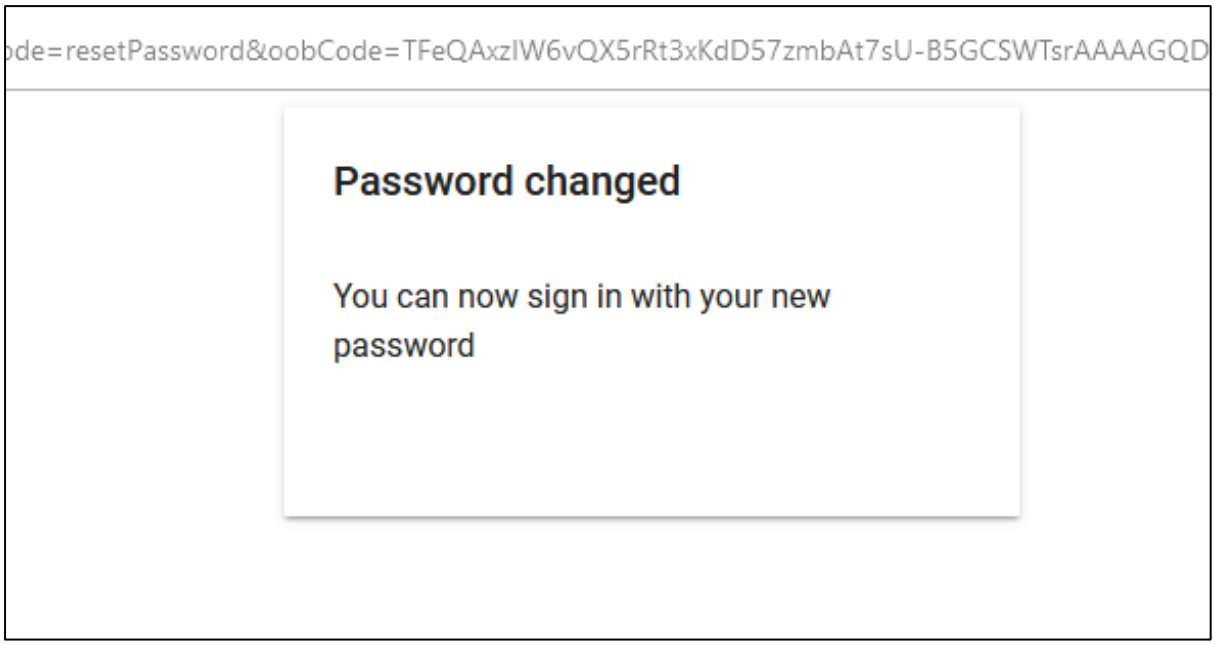
Slika 24. Poruka s poveznicom za resetiranje lozinke

U poruci za resetiranje lozinke, navedena je email adresa korisničkog računa za koji se zatražila promjena lozinke, koja je ujedno i email adresa na koju dolazi poruka (ne može se resetirati lozinka tako da se poruka šalje na neku drugu, nepovezanu email adresu).



Slika 25. Zaslon za promjenu lozinke

Kada korisnik klikne na poveznicu za promjenu lozinke, otvara se zaslon u web pregledniku u kojemu se nudi polje za unos nove lozinke. Nova lozinka mora, kao i u aplikaciji, poštivati pravilo Firebase-a, a to je da lozinka mora imati minimalno šest znakova.



Slika 26. Poruka o uspješno promijenjenoj lozinci

Kada korisnik unese ispravnu novu lozinku i stisne na tipku „Save“, prikazuje se poruka koja obavještava korisnika da je lozinka promijenjena te da se sada može prijaviti u aplikaciju s novom lozinkom.

4.3. ListView

`ListView` je najčešće korišteni *widget* u Flutteru za listanje zaslona. Prikazuje svoju „djecu“ jednu za drugom u smjeru pomicanja. Ako je dio zaslona na kojemu je implementiran `ListView` prazan, odnosno ako nema elemenata, tada će listanje biti onemogućeno kako ne bi došlo do nepredviđenog rada. U protivnom, ako postoji veći broj elemenata koji ispunjava

cijeli dio zaslona, tada je listanje omogućeno i korisnik se na tom dijelu zaslona može kretati gore-dolje ili lijevo-desno, ovisno o tome kako je definirano i implementirano.

U ovoj aplikaciji, ovaj *widget* je implementiran na glavnom zaslonu aplikacije, kod prikaza bilješki, korištenjem `ListView.builder` konstruktora.

```
return ListView.builder(
  itemCount: filteredNotes.length,
  itemBuilder: (BuildContext context, int index) {
    final DocumentSnapshot note = filteredNotes[index];

    return Container(
       …
    );
  });
```

Dva parametra koja se u ovom *widgetu* koriste su `itemCount` i `itemBuilder`. Prvi parametar je `itemCount`, koji uzima duljinu, odnosno broj bilješki koje se nalaze u listi. Ovaj parametar također uzima u obzir i sve promjene koje se događaju kod filtriranja, sortiranja i pretraživanja, o kojima će više biti rečeno u nadolazećim potpoglavljima. Drugi parametar je `itemBuilder` koji u funkciji vraća *widget* `Container` koji služi za vizualni prikaz svake bilješke, što je opisano u poglavlju o glavnom zaslonu aplikacije.

4.3.1 Filtriranje

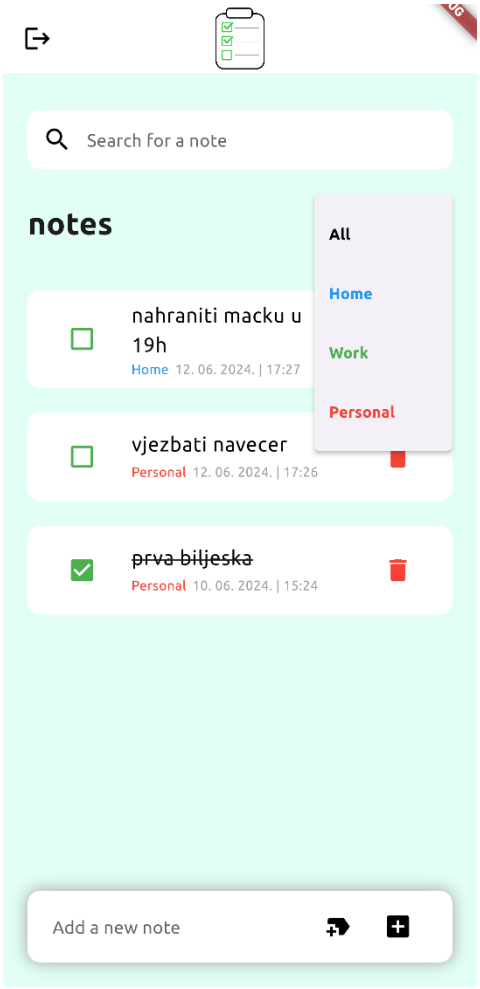
```
PopupMenuButton<String>(
  icon: const Icon(Icons.label),
  iconColor: Colors.black,
  color: Colors.white,
  itemBuilder: (BuildContext context) {
    return [
      PopupMenuItem<String>(
        value: 'All',
        child: Row(
          children: [
            Text('All',
                style: GoogleFonts.ubuntu(
                    color: Colors.black,
                    fontWeight: FontWeight.bold)),
          ],
        ),
      ),
      ...availableTags.map(
        (tag) {
          return PopupMenuItem<String>(
            value: tag.name,
            child: Row(
              mainAxisAlignment:
                  MainAxisAlignment.spaceBetween,
              children: [
                Text(tag.name,
                    style: GoogleFonts.ubuntu(
                        color: findTagColor(
                            tag.name, availableTags),
                        fontWeight: FontWeight.bold)),
                if (filteringDropdownValue ==
                    tag.name)
                  const Icon(Icons.check, size: 16),
              ],
```

```
            ),
          );
        },
      ),
    ];
  },
  onSelected: (String? newValue) {
    setState(
      () {
        filteringDropdownValue = newValue ?? '';
      },
    );
  },
)
```

Tipka za filtriranje služi za filtriranje dodanih bilješki prema oznakama koje su im dodijeljene prilikom unosa. Oznake su navedene u sljedećoj listi:

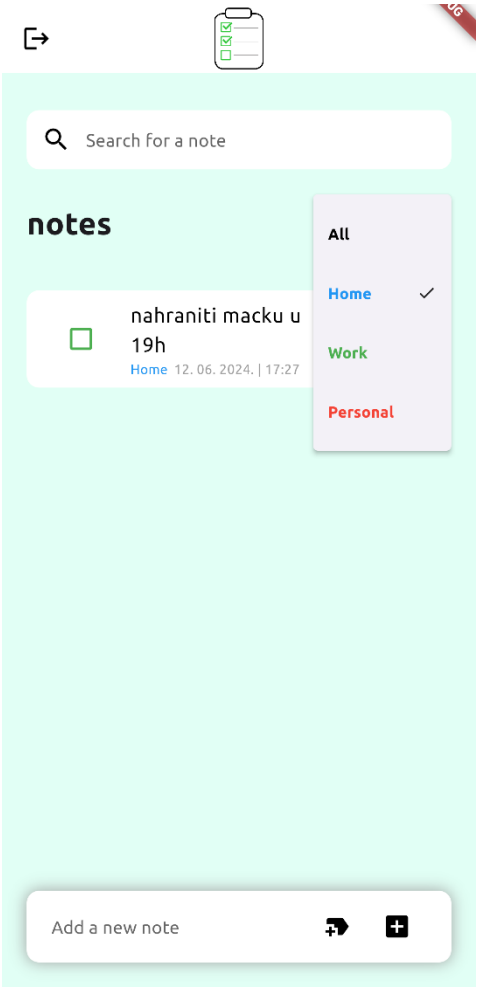
```
List<Tag> availableTags = [
  Tag(name: 'Home', color: Colors.blue),
  Tag(name: 'Work', color: Colors.green),
  Tag(name: 'Personal', color: Colors.red)
];
```

Postoje tri oznake: „Home”, „Work” i „Personal”. Svaka oznaka ima neku vlastitu boju, kako bi se oznake razlikovale ne samo po nazivu, već i po vizualnom aspektu. Svaka bilješka mora obavezno imati neku oznaku. Ako korisnik želi filtrirati svoje bilješke prema oznakama, to može učiniti tako da stisne na tipku za filtriranje. Ova tipka je kreirana uz pomoć *widgeta* `PopupMenuButton` koji služi za kreiranje padajućeg izbornika koji uzima vrijednosti iz liste s oznakama. U izborniku se osim navedenih vrijednosti iz liste nalazi i vrijednost „All“, koja je dodana pomoću widgeta `PopupMenuItem`. Ova vrijednost je ujedno i zadana vrijednost i koja prikazuje sve bilješke koje je korisnik unio, odnosno prikazuje bilješke koje imaju bilo koju oznaku.



Slika 27. Prikaz zadanog filtriranja bilješki

Kada je odabrana vrijednost „All“, u izborniku ne postoji oznaka kvačice koja to prikazuje, no ako, primjerice, korisnik odabere filtriranje prema oznaci “Home“, tada se pored vrijednosti „Home“ prikazuje kvačica.



Slika 28. Prikaz filtriranja bilješki s oznakom "Home"

4.3.2 Sortiranje

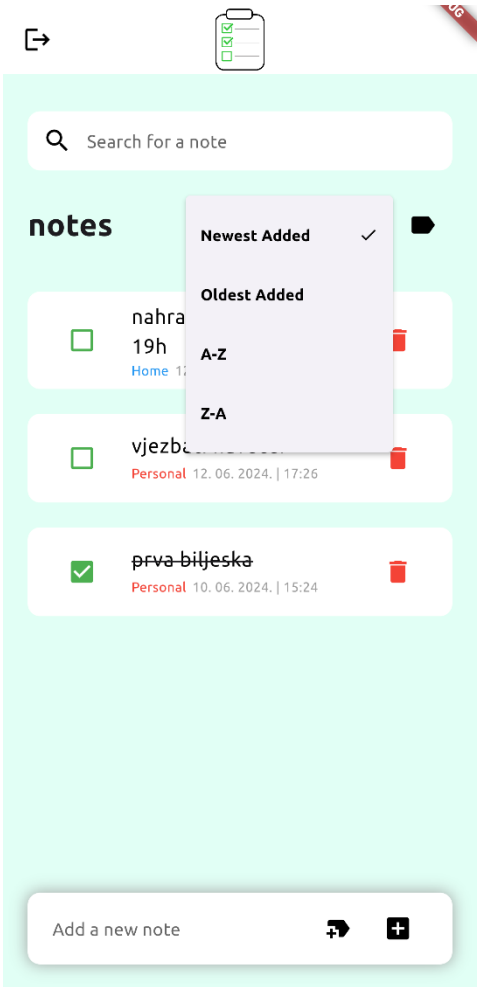
```
PopupMenuButton<String>(
  icon: const Icon(Icons.sort),
  iconColor: Colors.black,
  color: Colors.white,
  itemBuilder: (BuildContext context) {
    return sortingOptions.map((option) {
      return PopupMenuItem<String>(
        value: option['value'],
        child: Row(
          mainAxisAlignment:
              MainAxisAlignment.spaceBetween,
          children: [
            Text(
              option['label']!,
              style: GoogleFonts.ubuntu(
                  color: Colors.black,
                  fontWeight: FontWeight.bold),
            ),
            if (sortingDropdownValue ==
                option['value'])
              const Icon(Icons.check,
                  size: 16, color: Colors.black),
          ],
        ),
```

```
        );
      }).toList();
    },
    onSelected: (String? newValue) {
      setState(() {
        sortingDropdownValue = newValue!;
      });
    },
)
```

Tipka za sortiranje kreirana je korištenjem *widgeta* `PopupMenuButton` koji služi za kreiranje padajućeg izbornika. Padajući izbornik uzima vrijednosti iz sljedeće liste:

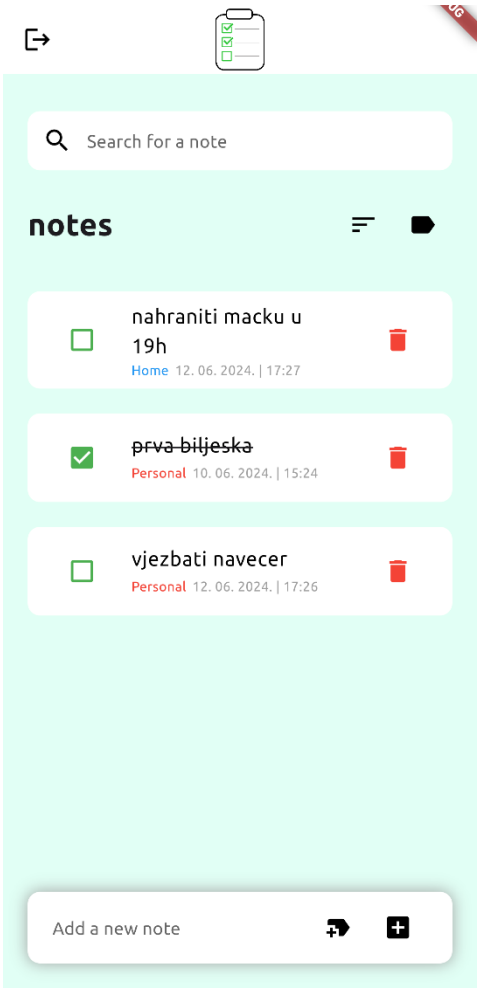
```
List<Map<String, String>> sortingOptions = [
  {'value': 'newest', 'label': 'Newest Added'},
  {'value': 'oldest', 'label': 'Oldest Added'},
  {'value': 'az', 'label': 'A-Z'},
  {'value': 'za', 'label': 'Z-A'},
];
```

U listi se nalaze četiri opcije za sortiranje: prema najnovije dodanim bilješkama, prema najstarije dodanim bilješkama te prema abecednom redoslijedu uzlazno i silazno (od A do Z te od Z prema A). Početna vrijednost je sortiranje prema najnovije dodanim bilješkama.



Slika 29. Prikaz padajućeg izbornika s odabranom početnom vrijednosti

Kada je neka vrijednost odabrana u padajućem izborniku, desno od te vrijednosti se nalazi kvačica kako bi korisnik mogao provjeriti prema čemu se bilješke trenutno sortiraju. Sortiranje bilješki radi i u kombinaciji s filtriranjem i pretraživanjem.

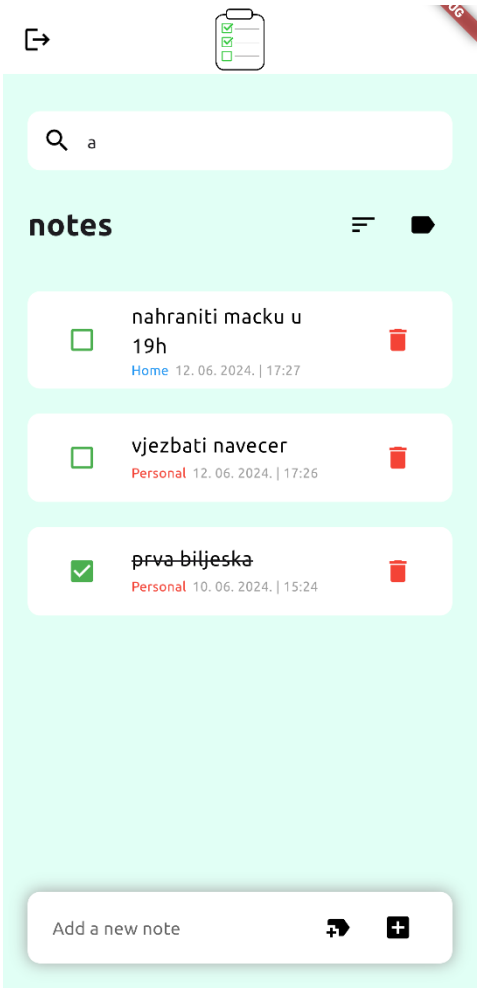


Slika 30. Prikaz sortiranja bilješki abecednim redoslijedom uzlazno

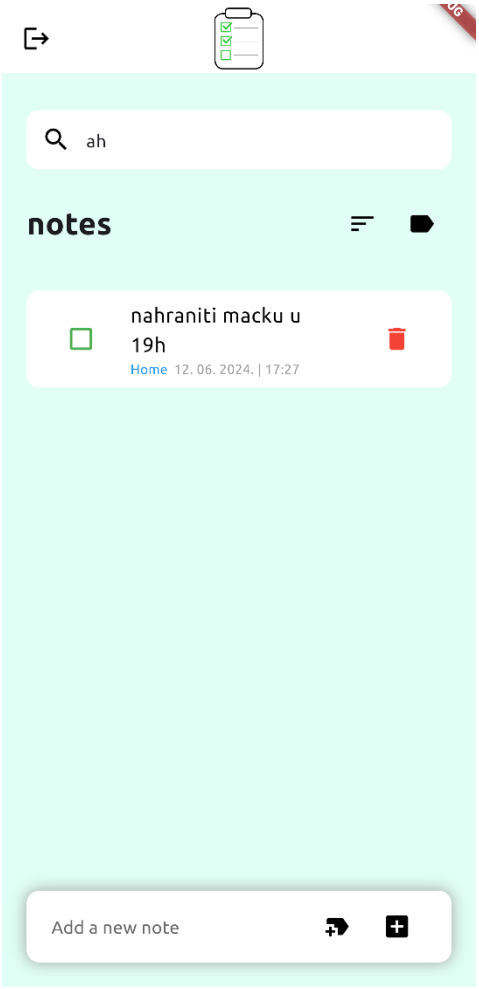
4.3.3 Pretraživanje

```
Container(
  decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    onChanged: (value) {
      setState(() { searchText = value.toLowerCase(); });
    },
    controller: searchController,
    style: GoogleFonts.ubuntu(),
    decoration: InputDecoration(
        prefixIcon: const Icon(Icons.search, color: Colors.black),
        border: InputBorder.none,
        hintText: 'Search for a note',
        hintStyle: GoogleFonts.ubuntu()),
  ),
)
```

Nakon alatne trake na vrhu zaslona, na zaslonu se nalazi polje za unos koje služi za pretraživanje bilješki. Polje je napravljeno korištenjem *widgeta* `Container` koji kao „dijete“ ima *widget* `TextField`. Polje je bijele boje, a u sebi sadrži tekst „Search for a note“ koji označava čemu služi ovo polje. Kada se krene unositi tekst, font je crne boje te se prilikom unosa svakog znaka u sučelju filtriraju bilješke koje sadrže uneseni tekst.



Slika 31. Prikaz rezultata pretraživanja unosom slova "a"



Slika 32. Prikaz rezultata pretraživanja unosom riječi "ah"

Kao što je prikazano na slici iznad, prikaz bilješki u sučelju filtrira i prikazuje sve bilješke koje odgovaraju unesenim riječima, odnosno koje sadrže unesene riječi negdje unutar svog teksta. Pretraživanje se također može kombinirati uz sortiranje i filtriranje prema oznakama.

4.4. Unos i forme

U ovom poglavlju će detaljno biti opisana implementacija polja za unos teksta, kao što su polja za unos email adrese i lozinke kod autentifikacije korisnika, te implementacija polja za unos bilješki na glavnom zaslonu aplikacije.

4.4.1 Polja za unos email adrese i lozinke

Polje za unos email adrese koristi Stateless *widget*.

```
class MailTextfield extends StatelessWidget {
  final TextEditingController controller;

  const MailTextfield({super.key, required this.controller});

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 50),
```

```
      child: TextField(
        controller: controller,
        keyboardType: TextInputType.emailAddress,
        style: GoogleFonts.ubuntu(),
        decoration: InputDecoration(
          enabledBorder: const OutlineInputBorder(
              borderSide: BorderSide(color: Colors.white)),
          focusedBorder: const OutlineInputBorder(
              borderSide: BorderSide(color: Colors.black)),
          fillColor: Colors.white,
          filled: true,
          hintText: 'Email',
          hintStyle: GoogleFonts.ubuntu(),
        ),
      ),
    );
  }
}
```

Na početku je definiran kontroler za unos teksta koji je potreban kako bi unos uopće bio moguć. Unutar *widgeta* `Padding` nalazi se *widget* `TextField` koji prihvaća kontroler za unos. S obzirom da se ova klasa poziva u datotekama koje su vezane uz zaslone za registraciju, prijavu i resetiranje lozinke, koriste se kontroleri koji su definirani u tim datotekama.

Tipkovnica koja se nudi korisniku prilikom unosa u ovom polju je tipkovnica namijenjena upravo za unos email adrese. Ova vrsta tipkovnice pored tipke razmaka umjesto simbola zareza nudi simbol @. Ako je ovo polje u fokusu, odnosno ako je odabrano, rub polja postaje crne boje kako bi što označava da je ovo polje odabrano te da se u ovom polju trenutno vrši unos. Ako nije, polje ima bijeli rub, što se poklapa uz boju samog polja koja je također bijela.

```
class _PasswordInputState extends State<PasswordInput> {
  bool obscuredPassword = true;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 50),
      child: TextFormField(
        controller: widget.controller,
        obscureText: obscuredPassword,
        style: GoogleFonts.ubuntu(),
        decoration: InputDecoration(
          enabledBorder: const OutlineInputBorder(
              borderSide: BorderSide(color: Colors.white)),
          focusedBorder: const OutlineInputBorder(
              borderSide: BorderSide(color: Colors.black)),
          fillColor: Colors.white,
          filled: true,
          hintText: 'Password',
          hintStyle: GoogleFonts.ubuntu(),
          suffixIcon: IconButton(
            icon: obscuredPassword
                ? const Icon(Icons.visibility)
                : const Icon(Icons.visibility_off),
            onPressed: () {
              setState(() {
                obscuredPassword = !obscuredPassword;
              });
            },
```

```
          ),
        ),
      ),
    );
  }
}
```

Za razliku od polja za unos email adrese, kod polja za unos lozinke korišten je Stateful *widget* iz razloga što se u ovom polju nalazi element prikazivanja i skrivanja unosa lozinke. Što se tiče vizualnih elemenata ovog polja, izgled i ponašanje je jednako kao i kod polja za unos email adrese. Jedina razlika je upravo navedeni element za prikaz/skrivanje lozinke. Ovaj element je implementiran pomoću tipke koja se nalazi na desnoj strani polja, s ikonom oka te uz korištenje varijable `obscuredPassword`. Početna vrijednost varijable `obscuredPassword` postavljena je na `true`, što znači da se lozinka prikazuje kao šifrirana. Kada korisnik stisne na ovu tipku, vrijednost varijable se mijenja na `false`, lozinka se prikazuje i mijenja se ikona tipke.

4.4.2 Polje za unos bilješki

```
Container(
  padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 5),
  margin: const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
  decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(10),
    boxShadow: const [
      BoxShadow(color: Colors.grey, blurRadius: 10.0)
    ],
  ),
  child: TextField(
    minLines: 1,
    maxLines: 6,
    controller: noteController,
    style: GoogleFonts.ubuntu(),
    decoration: InputDecoration(
      hintText: 'Add a new note',
      hintStyle: GoogleFonts.ubuntu(),
      border: InputBorder.none,
      suffixIcon: Row(
        mainAxisSize: MainAxisSize.min,
        children: [
          IconButton(
            icon: const Icon(Icons.new_label),
            color: Colors.black,
            onPressed: () {
              showTagSelectionDialog(context);
            },
          ),
          IconButton(
            icon: const Icon(Icons.add_box),
            color: Colors.black,
            onPressed: () async {
              String trimmedText = noteController.text.trim();

              if (trimmedText.isNotEmpty && selectedTag != null) {
                await DatabaseService(uid: uid)
                    .addNote(trimmedText, selectedTag!, false);
                noteController.clear();
```

```
                    selectedTag = null;
                  }
                },
              ),
            ],
          ),
        ),
      ),
    )
```

Polje za unos teksta bilješki „omotano“ je *widgetom* `Container`, što je slučaj kao i kod prikaza bilješki u sučelju. To je napravljeno u kako bi se pored samog polja za unos mogle prikazati tipke za odabir oznake te za unos bilješke. *Widget* `Container` je bijele boje, ima nekoliko modifikacija oko margina i odmaka od rubova, a uz to ima i blagu sjenu sive boje, s obzirom da se ovaj element sučelja prikazuje ispred ostalih kada se otvori tipkovnica za unos.

Što se tiče samog unosa, kao i kod prethodnih polja za unos, korišten je widget `TextField`. Postavljeni su i parametri `minLines` i `maxLines` koji su zaduženi da se, kada se vrši unos teksta bilješke, polje za unos povećava maksimalno do unesenih šest redova teksta, a nakon toga se više ne povećava, nego se omogućuje listanje ovog polja. Pored polja za unos nalazi se tipka za odabir oznaka, koja, kada korisnik stisne na istu, prikazuje izbornik koji je prikazan na slici 12.

Ako postoji uneseni tekst, odnosno ako nije prazno polje ili ako nisu samo razmaci dodani, te ako je odabrana oznaka za bilješku i korisnik tada stisne na tipku za unos, bilješka se uspješno dodaje u Firestore bazu podataka kao i u sučelje, a polje za unos teksta se čisti te više niti jedna oznaka nije odabrana, kako bi se odmah omogućio sljedeći unos. Ako jedan od ova dva uvjeta ili ako oba uvjeta nisu ispunjena, unos bilješke je neuspješan te se ispisuje jedna od poruka unutar *widgeta* `SnackBar` (slika 16).

5. Zaključak

U današnjem digitalnom dobu, mobilne aplikacije su ključne za efikasno obavljanje svakodnevnih aktivnosti. Od praćenja poslovnih ili studentskih obaveza pa do upravljanja vlastitim financijama i internetske trgovine, mnoge mobilne aplikacije nastoje korisnicima olakšati svakodnevni život.

Kroz razvoj mobilne aplikacije „you to-do“ za upravljanje bilješkama, istražene su i implementirane funkcionalnosti koje korisnicima olakšavaju praćenje svojih obaveza vezanih uz posao, studiranje ili uz svakodnevni život. Aplikacija nudi korisnicima jednostavno sučelje za dodavanje bilješki, označavanje bilješki kao dovršenih te brisanje istih, integrirajući funkcionalnosti pretraživanja, sortiranja prema abecednom redoslijedu ili vremenu dodavanja, te filtriranja prema oznakama. Ovaj rad ističe važnost tehnoloških rješenja u organizaciji svakodnevnih osobnih i poslovnih obaveza, s fokusom na poboljšanje produktivnosti i efikasnosti.

Literatura

[1] S. Alessandria, Flutter Cookbook, Birmingham: Packt Publishing Ltd., 2023.

[2] D. Joshi, Building Cross-Platform Apps with Flutter and Dart, London: BPB Online, 2023.

[3] »Flutter Documentation - Navigation,« Google, [Mrežno]. Available: https://docs.flutter.dev/ui/navigation. [Pokušaj pristupa 10. Lipanj 2024.].

[4] »Flutter Documentation - State management,« Google, [Mrežno]. Available: https://docs.flutter.dev/get-started/fwe/state-management. [Pokušaj pristupa 12. June 2024.].

[5] »Flutter API Documentation - initState method,« Google, [Mrežno]. Available: https://api.flutter.dev/flutter/widgets/State/initState.html. [Pokušaj pristupa 12. Lipanj 2024.].

[6] »Flutter API Documentation - dispose method,« Google, [Mrežno]. Available: https://api.flutter.dev/flutter/widgets/State/dispose.html. [Pokušaj pristupa 12. Lipanj 2024.].

[7] »Flutter API Documentation - build method,« Google, [Mrežno]. Available: https://api.flutter.dev/flutter/widgets/StatelessWidget/build.html. [Pokušaj pristupa 12. Lipanj 2024.].

[8] »Flutter API Documentation - setState method,« Google, [Mrežno]. Available: https://api.flutter.dev/flutter/widgets/State/setState.html. [Pokušaj pristupa 12. Lipanj 2024.].

[9] »Flutter Documentation - Create and style a text field,« Google, [Mrežno]. Available: https://docs.flutter.dev/cookbook/forms/text-input. [Pokušaj pristupa 29. Lipanj 2024.].

[10] »Flutter Documentation - Retrieve the value of a text field,« Google, [Mrežno]. Available: https://docs.flutter.dev/cookbook/forms/retrieve-input. [Pokušaj pristupa 29. Lipanj 2024.].

Popis slika