Razvoj aplikacije za čavrljanje

Bedenic, Dominik

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Preddiplomski studij Informatika

Dominik Bedenic

Razvoj aplikacije za čavrljanje

Završni rad

Mentor: dr. sc. Marija Brkić Bakarić

Rijeka, 01.09.2023.

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 26.5.2023.

Zadatak za završni rad

Pristupnik: Dominik Bedenic

Naziv završnog rada: Razvoj aplikacije za čavrljanje

Naziv završnog rada na engleskom jeziku: Development of chat application

Sadržaj zadatka:

Cilj ovog zadatka je dizajnirati i programirati aplikaciju za čavrljanje korištenjem Flutter okvira za razvoj aplikacija. Glavni naglasak je na stvaranju intuitivnog korisničkog sučelja i funkcionalnosti za olakšanu razmjenu poruka. U okviru rada, osim same izrade aplikacije, detaljno su opisani i objašnjeni svi aspekti aplikacije. To uključuje tehničke detalje, poput odabranih alata, biblioteka i tehnologija koje su korištene za izgradnju aplikacije.

Mentor

Izv. prof. dr. sc. Marija Brkić Bakarić

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 26.5.2023.

Dominik Bedenic

(potpis pristupnika)

Adresa Radmile Matejčić 2
51000Rijeka, Hrvatska
Tel: +385(0)51 584 700
E-mail: ured@inf.uniri.hr
OIB: 64218323816
IBAN: HR1524020061400006966

Sadržaj

Sažetak

Svrha ovog završnog rada je predstaviti sveobuhvatan pregled procesa razvoja i implementacije aplikacije za čavrljanje koja koristi okvir *Flutter* za frontend[1] i *Firebase* kao backend. Cilj je razviti mobilnu komunikacijsku platformu koja je jednostavna za korištenje.

U uvodu rada naglašava se važnost mobilnih aplikacija u sadašnjoj digitalnoj eri, kao i potražnja za učinkovitim i sigurnim komunikacijskim alatima. Također je naglašena vrijednost *Firebase-a* kao skalabilne i pouzdane pozadinske usluge i *Flutter-a* kao razvojnog okvira za višeplatformski razvoj aplikacija.

Nakon toga slijedi poglavlje s metodologijom istraživanja koja ocrtava radnje poduzete za postizanje ciljeva projekta. Poglavlje pokriva početnu fazu planiranja, koja uključuje određivanje ciljeva i specifikacija, stvaranje korisničkog sučelja i definiranje strukture baze podataka.

U radu se zatim raspravlja o glavnim elementima i značajkama aplikacije prije nego što se uđe u specifičnosti implementacije. To uključuje izgradnju grupa za razgovor, razmjenu poruka i ažuriranja u stvarnom vremenu. Detaljno je opisano kako iskoristiti *Firebase Firestore* bazu podataka i *Firebase Authentication* za rukovanje registracijom korisnika.

Implementacijski detalji i izgledi glavne su teme posljednjeg poglavlja. Ističe uspješnu izradu upotrebljive aplikacije za razgovor.

Ovaj završni rad naglašava važnost tehnologija za razvoj suvremenih mobilnih aplikacija.

Ključne riječi: aplikacija za čavrljanje, *Flutter*, *Firebase*, mobilni razvoj, komunikacija u stvarnom vremenu, backend, više platformi, autentifikacija korisnika, grupa za razgovore, razmjena poruka, *Firestore* baza podataka, skalabilnost, sigurnost, *Figma*, UI, korisničko sučelje.

[1] Korisničko sučelje koje omogućuje korisnicima da vide sadržaj, izvršavaju akcije i komuniciraju s aplikacijom.

1. Uvod

1.1. Ideja i ciljevi aplikacije

Glavni cilj ovog rada bio je proizvesti mobilnu aplikaciju s mnoštvom značajki koje omogućuju povezivanje korisnika i njihov međusoban razgovor u stvarnom vremenu. Glavni ciljevi bili su osigurati besprijekorno korisničko iskustvo, osigurati autentifikaciju, omogućiti stvaranje i otkrivanje *chat* grupa te omogućiti komunikaciju između korisnika unutar grupe.

1.1.1. Autentifikacija

Autentifikacija korisnika bila je jedna od temeljnih komponenti aplikacije. Korisnici mogu stvarati račune i sigurno pristupiti aplikaciji zahvaljujući snažnom sustavu provjere autentičnosti. Korisnici moraju unijeti svoje vjerodajnice na određenu stranicu za registraciju, a nakon provjere valjanosti dobivaju pristup svojoj individualnoj nadzornoj ploči.

1.1.2. Stvaranje i otkrivanje grupa

Aplikacija omogućuje korisnicima da izgrade grupe za razgovor kako bi bili u učinkovitoj komunikaciji i suradnji. Korisnici mogu razgovarati, razmjenjivati ideje i raditi zajedno na određenim temama ili interesima unutar ovih grupa. Korisnici mogu definirati naziv za nove grupe za razgovor preko korisničkog sučelja aplikacije.

1.1.3. Čavrljanje u stvarnom vremenu

Korisnici koji su se pridružili *chat* grupi mogu komunicirati s ostalim članovima grupe u stvarnom vremenu. Kako bi se osigurala brza sinkronizacija i isporuka poruka svim članovima grupe, aplikacija koristi snagu *Firebase-ove* baze podataka.

1.1.4. UI dizajn

Estetski dizajn aplikacije bio je još jedna ključna komponenta. Aplikacija koristi *Flutter-ovu* opsežnu biblioteku elemenata korisničkog sučelja i widgeta za stvaranje estetski ugodnog dizajna koji je poboljšao cjelokupno korisničko iskustvo, učinio aplikaciju vizualno privlačnom i jednostavnom za korištenje. Kod procesa dizajniranja korisničkog sučelja koristio se alat *Figma*[2].

1.2. Flutter

Flutter je postao moćan i prilagodljiv okvir u svijetu razvoja mobilnih aplikacija koji se stalno mijenja. *Flutter*, kojeg je razvio Google, iznimno je voljen među programerima zbog svoje sposobnosti stvaranja zapanjujućih i učinkovitih aplikacija na nekoliko platformi. Uz pomoć *Flutter-a*, UI alata otvorenog koda[3], programeri mogu proizvesti desktop, web i mobilne aplikacije iz jedne baze koda. Koristi se reaktivno programiranje, pružajući brzo i fluidno korisničko iskustvo automatskim ažuriranjem korisničkog sučelja. *Flutter-ov* naglasak na razvoju estetski privlačnih korisničkih sučelja jedan je od njegovih najbitnijih aspekata. *Flutter* omogućuje programerima da jednostavno kreiraju prekrasna korisnička sučelja koja izgledaju dobro na svim platformama zahvaljujući vlastitom skupu prilagodljivih widgeta i velikom izboru unaprijed dizajniranih komponenti. Mogućnost rada *Flutter* aplikacija

[2] Suradnička web aplikacija za dizajn sučelja.
[3] Odnosi se na softver čiji je izvorni kod dostupan u okviru open source licence svim korisnicima koji mogu mijenjati, prepravljati i unaprjeđivati njegov sadržaj.

na različitim platformama velika je prednost. Programeri mogu istovremeno ciljati na brojne platforme, uključujući iOS, Android, web, pa čak i računala, korištenjem jedne baze koda. Također osigurava jednako ponašanje i izgled na različitim uređajima i operativnim sustavima, što pojednostavljuje proces razvoja. *Flutter-ove* performanse su također impresivne. *Flutter* koristi vlastiti mehanizam za renderiranje kako bi zaobišao izvorne komponente platforme i izravno generirao svoje korisničko sučelje, proizvodeći munjevito brze performanse i fluidne animacije. Dodatno, *Flutter-ova* funkcija „Hot Reload“ ubrzava razvoj omogućujući programerima da odmah procijene promjene u svom kodu bez potrebe za izvođenjem ponovne izgradnje. Osim toga, *Firebase*, Googleova opsežna zbirka pozadinskih usluga, lako se integrira s *Flutter-om*, dajući razvojnim programerima pristup snažnoj backend[4] infrastrukturi koja uključuje autentifikaciju korisnika, baze podataka u stvarnom vremenu, pohranu u oblaku i još mnogo toga. Ova integracija olakšava stvaranje pouzdanih i skalabilnih aplikacija. *Flutter* je transformirao proces stvaranja mobilnih aplikacija pružajući robustan okvir za više platformi koji daje programerima resurse koji su im potrebni za izradu atraktivnih korisničkih sučelja i pružanje vrhunskih korisničkih iskustava zahvaljujući ogromnoj biblioteci widgeta[5], velikoj brzini i integraciji s *Firebase-om*. Flutter donosi nove prilike za stvaranje aplikacija koje se ističu na današnjem konkurentnom digitalnom tržištu.

[4] Dio aplikacije koji nije vidljiv korisnicima, ali je ključan za njeno ispravno funkcioniranje.
[5] Osnovne građevne jedinice u Flutter okviru za izradu korisničkih sučelja.

1.3. Firebase

U području razvoja suvremenih aplikacija, *Firebase* je poznat kao snažna i sveobuhvatna backend platforma koja programerima olakšava stvaranje skalabilnih aplikacija. Usluge i alati temeljeni na oblaku koje nudi Google *Firebase* pojednostavljuju razvojni proces i omogućuju programerima da se koncentriraju na stvaranje izvanrednih korisničkih iskustava. Autentifikacija, baze podataka u stvarnom vremenu, pohrana u oblaku, hosting, strojno učenje i druge usluge uključene su u objedinjenu platformu. Koristeći *Firebase*, programeri mogu jednostavno uključiti ove usluge u svoje aplikacije, uklanjajući potrebu za izgradnjom i održavanjem komplicirane backend infrastrukture. Jedna od glavnih prednosti *Firebase-a* je njegova baza podataka u stvarnom vremenu (eng. *realtime database*), koja programerima omogućuje stvaranje reaktivnih aplikacija. *Firebase-ova* baza podataka u stvarnom vremenu, koju podržava Google-ova NoSQL[6] tehnologija baze podataka, omogućuje sinkronizirane, trenutne promjene podataka na brojnim uređajima. Osim toga, *Firebase* pruža jake usluge provjere autentičnosti koje programerima olakšavaju uspostavljanje sigurnih procesa autorizacije korisnika. Osiguravajući zaštitu i integritet korisničkih podataka, *Firebase* nudi besprijekorno i sigurno iskustvo autentifikacije korisnika s podrškom za e-poštu/lozinku, prijavu pomoću društvenih mreža i prilagođene metode autentifikacije. Usluga pohrane u oblaku koju nudi je još jedna moćna značajka. Koristeći *Firebase-ovu* skalabilnu i sigurnu arhitekturu pohrane u oblaku, programeri mogu jednostavno pohraniti i dohvatiti materijale koje su generirali korisnici, uključujući fotografije, videozapise i dokumente. To omogućuje dijeljenje sadržaja, timski rad i učinkovito upravljanje podacima unutar aplikacija. Osim toga, *Firebase* lako komunicira s dobro poznatim programskim okvirima kao što su *Flutter*, *Angular* i *React Native*, dajući programerima izvrsno razvojno iskustvo. Kao rezultat toga, programeri mogu iskoristiti robusne značajke i usluge *Firebase-a* bez obzira na razvojno okruženje koje preferiraju. Zaključno, *Firebase* je postao vrhunska pozadinska platforma koja programerima omogućuje lako stvaranje skalabilnih aplikacija bogatih značajkama i aplikacija u stvarnom vremenu.

[6] Termin koji se koristi za opisivanje baze podataka koje ne koriste tradicionalni relacijski model kao što to čine SQL baze podataka.

2. Dizajn

2.1. Frontend dizajn

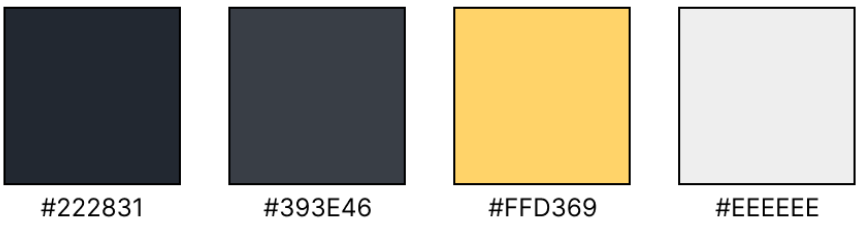
Zanimljivom i estetski ugodnom korisničkom iskustvu uvelike je pridonio dizajn korisničkog sučelja. Procedura je uključivala prikupljanje ideja iz već postojećih dizajna aplikacija za čavrljanje, pažljivo biranje palete boja i sastavljanje dizajna korištenjem popularnog programa za dizajn, *Figme*.

2.1.1. Traženje inspiracije

Provedeno je opsežno istraživanje kako bi se analizirao dizajn usporedivih aplikacija za čavrljanje koje su bile dostupne na webu. Ovim su se istraživanjem tražile ideje za prepoznatljivim i estetski ugodnim sučeljem, dok su se također identificirali uobičajeni obrasci, trendovi i najbolje prakse u dizajnu aplikacija.

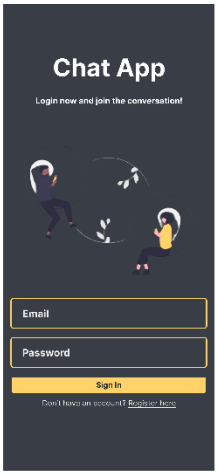
2.1.2. Biranje palete boja

Sljedeća faza bila je odabir prikladne palete boja koja bi odgovarala općoj temi i raspoloženju aplikacije nakon prikupljanja uvida iz trenutnog dizajna.

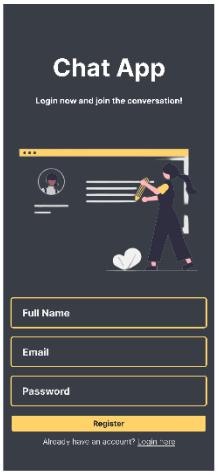


Slika 1 Odabrana paleta boja

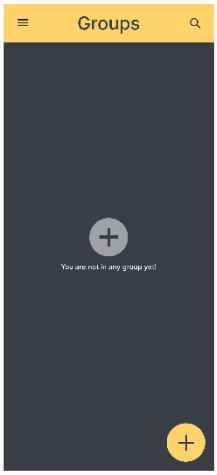
Kako bi se stvorio privlačan kontrast s primarnom bojom, boja sa heksadecimalnim kodom #393E46 odabrana je kao pozadinska boja. Ova pomoćna boja pomogla je u razlikovanju različitih komponenti korisničkog sučelja. Boja sa heksadecimalnim kodom #FFD369 korištena je kao primarna boja za poboljšanje svjetline i vizualne intrige. Ova nijansa dala je programu malo života i korištena je kao boja za isticanje gumba, obavijesti i drugih značajnih komponenti. Čist i nenaglašen kontrast između primarne i sekundarne boje dodan je korištenjem boje teksta heksadecimalnog koda #EEEEEE. Kako bi se naglasili određeni elementi teksta bez nadjačavanja dizajna u cjelini, ova boja teksta također je korištena za naslove, podnaslove i druge elemente teksta.



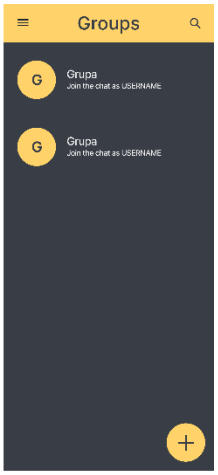
Stranica za prijavu



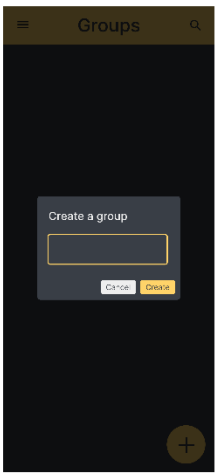
Stranica za registraciju



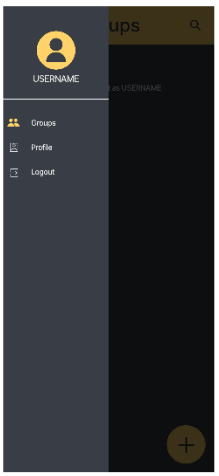
Početna stranica bez grupa



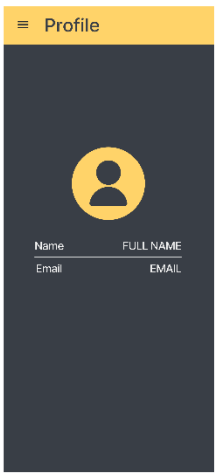
Početna stranica sa grupama



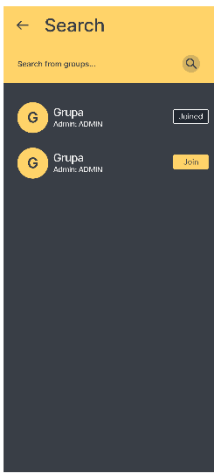
Kreiranje grupe



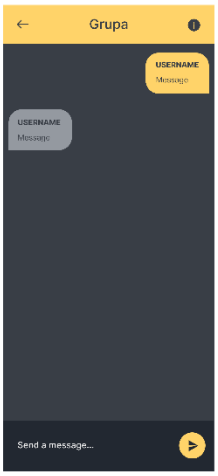
Glavni izbornik



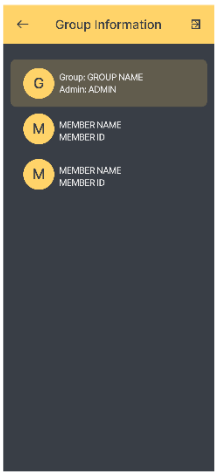
Stranica profila



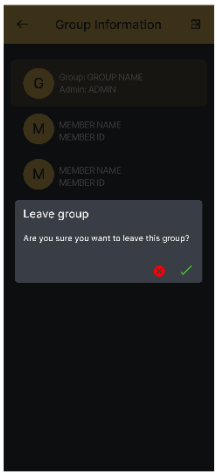
Stranica za traženje grupa



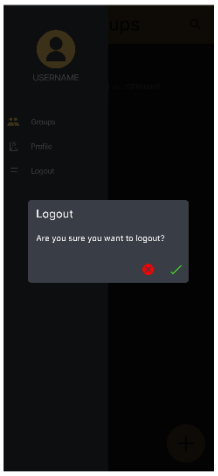
Stranica za čavrljanje



Stranica profila grupe



Izlazak iz grupe



Odjava iz aplikacije

Slika 2 Cjelokupan dizajn aplikacije

2.2. Backend dizajn

Backend dizajn aplikacije koristio je *Firebase Firestore* kao uslugu baze podataka za upravljanje i pohranu podataka. "Grupe" i "Korisnici" (eng. *„groups“, „members“*) su dvije osnovne zbirke koje čine strukturu baze podataka (Slika 3).

2.2.1. Kolekcija grupa („groups“)

Ova kolekcija služila je kao središnja lokacija za *chat* grupe aplikacije. Sljedeća polja bila su uključena u pojedinačne dokumente koje je predstavljala svaka grupa za razgovor:

- admin: ovo polje pohranjuje ID ili referencu na administratora grupe, omogućujući jednostavnu identifikaciju administratora grupe,
- groupId: jedinstveni identifikator za svaku grupu, koji služi kao primarni ključ za dokument grupe,
- groupName: naziv ili naslov dodijeljen *chat* grupi, pružajući jasan identifikator za korisnike,
- members: ovo polje pohranjuje niz korisnika koji su članovi grupe, olakšavajući upravljanje članstvom i operacije povezane s članovima,
- recentMessage: polje koje je pohranilo najnoviju poruku poslanu u grupi, omogućavajući brzi pristup najnovijoj aktivnosti unutar *chat-a*,
- recentMessageSender: ovo polje pohranjuje ID ili referencu na člana koji je poslao posljednju poruku, pomažući pri prikazu informacija o pošiljatelju uz poruku.

Svaki dokument grupe također je sadržavao ugniježđenu kolekciju pod nazivom "messages" ili poruke. Ti su dokumenti predstavljali pojedinačne komunikacije koje su poslane unutar grupe.

U svakom komunikacijskom dokumentu su prisutna sljedeća polja:

- message: tekstualna poruka poslana unutar *chat-a*, sam sadržaj poruke,
- sender: referenca na korisnika koji je poslao poruku, koja omogućuje jednostavnu identifikaciju pošiljatelja poruke,
- time: ovo polje pohranjuje vremensku oznaku kada je poruka poslana.

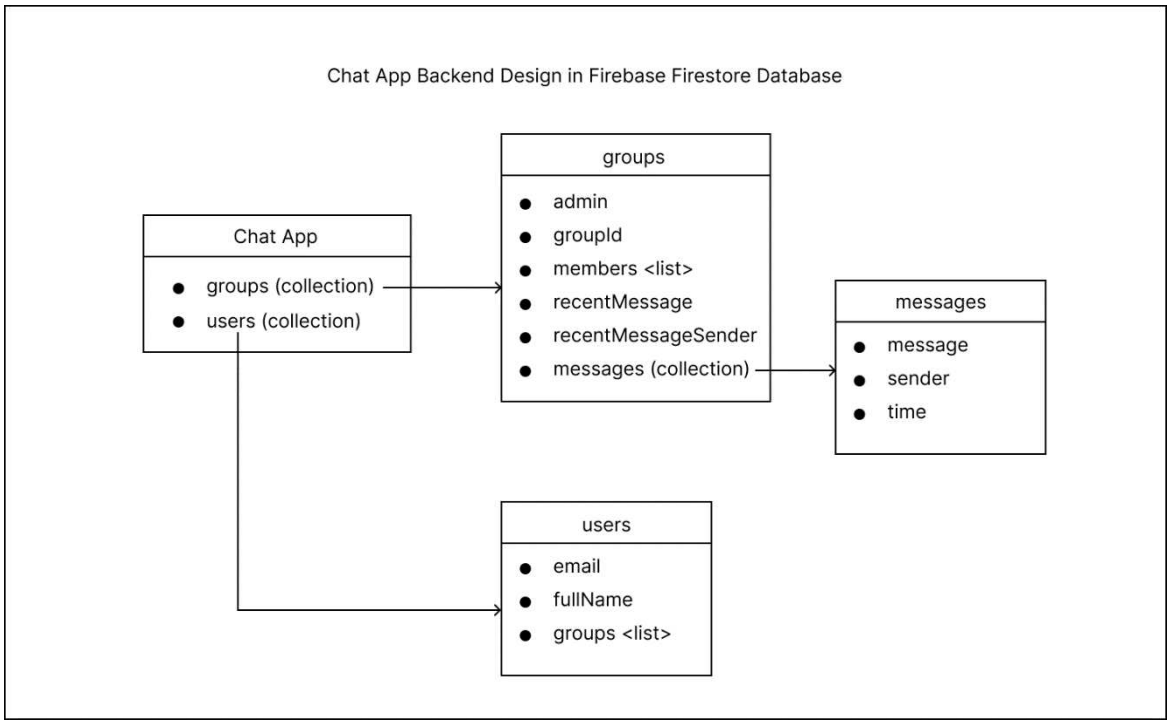
2.2.2. Kolekcija korisnika („users“)

Središnje mjesto za korisničke podatke je kolekcija "users". Unutar ove kolekcije postoji jedinstveni dokument za svakog korisnika u aplikaciji, koji je sadržavao polja prikazana u nastavku:

- email: adresa e-pošte povezana s korisničkim računom, koja služi kao jedinstveni identifikator,
- fullName: puno ime korisnika, pruža osobnu identifikaciju unutar aplikacije,
- groups: ovo polje pohranjuje niz referenci na *chat* grupe kojima se korisnik pridružio. Ovo polje olakšava praćenje članstva korisnika u grupama.

2.2.3. Analiza baze podataka

Korištenjem gore opisane strukture, backend dizajn ove aplikacije učinkovito je pohranio i organizirao podatke unutar *Firebase Firestore*. Kolekcije "groups" i "users", zajedno s ostalim povezanim poljima, omogućile su učinkovito upravljanje grupama za razgovor, korisnicima i podacima o porukama. Ova je struktura pružila čvrst temelj za implementaciju značajki kao što su stvaranje grupa, upravljanje članovima, pohrana poruka i provjera autentičnosti korisnika unutar aplikacije.



Slika 3 Vizualni prikaz baze podataka

3. Inicijalizacija projekta

Tijekom procesa razvoja *Flutter* aplikacije za čavrljanje, korištenje *Visual Studio Code* kao razvojnog okruženja (IDE) i *scrcpy* kao alata za zrcaljenje zaslona pokazalo se kao prednost, nudeći niz pogodnosti koje su poboljšale razvojno iskustvo i mogućnosti testiranja.

3.1. Razvojna okolina

3.1.1. Visual Studio Code

Visual Studio Code (*VS Code*) je razvojno okruženje (IDE[7]) koje se najčešće koristi za stvaranje *Flutter* aplikacija. Korištenje *VS code-a* bilo je korisno iz sljedećih razloga: *VS Code* nudi širok raspon dodataka, proširenja i značajki koje su posebno napravljene kako bi proces razvoja aplikacija bio učinkovitiji. Neki od dodataka koje *VS Code* nudi su ugrađeno otklanjanje pogrešaka, isticanje sintakse i dovršavanje koda. Opsežno tržište *VS Code-a* omogućuje programerima da biraju iz velikog izbora proširenja, omogućujući prilagodbu i integraciju dodatnih alata i funkcionalnosti. Ova fleksibilnost pomaže u ispunjavanju specifičnih projektnih zahtjeva i osobnih preferencija, povećavajući produktivnost i učinkovitost tijekom razvoja. Osim mnogih ekstenzija i dodatak *VS Code* dolazi sa integriranim terminalom, pomoću kojeg programeri mogu upravljati *Git* verzijama, pokretati skripte i pokretati *Flutter* naredbe bez napuštanja IDE-a. Uklanjanjem potrebe za prebacivanjem između različitih aplikacija, ovaj integrirani proces štedi vrijeme i trud. Ekstenzije *Flutter* i *Dart* su dodane kako bi se konfigurirao *VS Code* za razvoj *Flutter-a*. Ova su proširenja pružala funkcionalnost koja je olakšavala razvoj, uključujući isticanje sintakse, dovršavanje koda i mogućnosti otklanjanja pogrešaka. Osim navedenih ekstenzija instalirana je i ekstenzija „*Flutter Widget Snippets*“ koja nudi skup korisnih *Flutter* i *Dart* isječaka za brži razvoj aplikacije.

[7] Programska aplikacija koja pruža programerima sve potrebne alate i okruženje za razvoj softverskih aplikacija.

3.1.2. Scrcpy

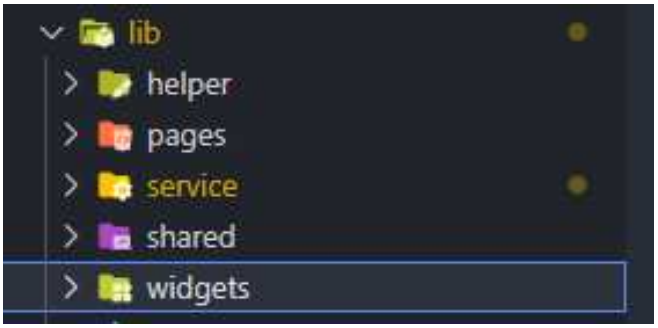
Program za zrcaljenje zaslona otvorenog koda *scrcpy* bio je vrlo zgodan kada se testirala aplikacija. Prilikom testiranja aplikacije, programeri je mogu pregledavati i komunicirati s njom na većem zaslonu zrcaljenjem zaslona mobilnog uređaja u stvarnom vremenu na zaslon računala. Uz pomoć ove funkcionalnosti moguće je uočiti vizualne probleme, testirati interakcije korisnika i osigurati da korisnici imaju isto iskustvo na svim uređajima. *Scrcpy* je poznat po svojim visokim performansama i minimalnom kašnjenju. Programeri mogu precizno ispitati funkcionalnost programa, pronaći bilo kakvo kašnjenje ili vizualne artefakte i poboljšati korisničko sučelje za besprijekorno korisničko iskustvo. *Scrcpy* je namijenjen jednostavnom postavljanju i korištenju. Programeri mogu odmah povezati svoje uređaje s računalom i započeti zrcaljenje zaslona uz samo nekoliko konfiguracijskih koraka.

3.2. Pokretanje projekta

Aplikacija je pokrenuta unosom potrebnih naredbi u sučelje naredbenog retka (CLI[8]). Da bi se to postiglo, trebalo je instalirati *Flutter SDK*, dodati binarne datoteke *Flutter* u PATH varijablu sustava i stvoriti novi *Flutter* projekt pomoću naredbe "flutter create". Osnovna struktura projekta napravljena je definiranjem naziva projekta i odabirom željene platforme (moj odabir su *Android* i web). Zatim sam nastavio s definiranjem strukture aplikacije. Osnovna struktura aplikacije je slijedeća:

1. Folder „helper“ koji sadrži varijable za ključeve i metode za spremanje i dobivanje podataka iz aplikacije.
2. Folder „pages“ koji sadrži skup svih korisničkih sučelja aplikacije.
3. Folder „service“ koji sadrži skup svih metoda za rad aplikacije i Firebase-a.
4. Folder „shared“ koji sadrži konstante kao što su boje i informacije o projektu.
5. Folder „widgets“ koji sadrži sve prilagođene widgete, na primjer za izgled grupnih pločica ili oblačića za čavrljanje.

[8] Način interakcije s računalnim programom ili sustavom putem naredbenog retka, bez potrebe za grafičkim sučeljem.



Slika 4 Struktura aplikacije u programu VS Code

3.3. Povezivanje aplikacije s backendom

Za uspostavljanje veze između vaše *Flutter chat* aplikacije i *Firebase-a* poduzeto je nekoliko koraka, počevši od postavljanja projekta na *Firebase* konzoli do konfiguracije i integracije *Firebase* usluga unutar aplikacije. Na *Firebase* konzoli prvo je kreiran novi projekt. Novi *Firebase* projekt koji je kreiran posebno za aplikaciju za čavrljanje nastao je unosom potrebnih informacija i odabirom željenih opcija. Tim je postupkom stvoren poseban ID projekta i omogućeno korištenje različitih *Firebase* usluga. Sljedeći korak uključivao je dobivanje datoteke "google-services.json". Ova datoteka djeluje kao most između aplikacije i *Firebase* usluga i pruža kritične podatke o konfiguraciji posebno za projekt. Preuzeta je s Firebase konzole i nakon toga dodana u direktorij "android/app“.

Slika 5 Sastav google-services.json datoteke

Firestore i *Firebase Authentication* bili su uključeni za projekt na *Firebase* konzoli. Baza podataka koja se koristi za pohranjivanje podataka grupa za *chat* i poruka je *Firestore*, a za prijavu i registraciju korisnika koriste se metode provjere autentičnosti temeljene na e-pošti i lozinki. Zadana sigurnosna pravila promijenjena su kako bi se omogućio siguran pristup *Firestore-u*. Samo autentificirani korisnici sada imaju pristup za čitanje i pisanje zbog revidiranih sigurnosnih ograničenja. Program je dobio dodatni sloj sigurnosti kao rezultat novih sigurnosnih pravila, koja su osigurala da samo ovlašteni korisnici mogu pristupiti i uređivati *Firestore* podatke.

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth!=null;
    }
  }
}
```

Slika 6 Sigurnosna pravila u Firestore-u

Projektu su dodane potrebne ovisnosti kako bi se *Firebase* usluge integrirale u *Flutter* aplikaciju. Redak „implementation platform('com.google.firebase:firebase-bom:32.1.0')“ bio je prisutan u odjeljku „dependencies“ unutar „app/build.gradle“ datoteke. Putanja klase „com.google.gms:google-services:4.3.15“ dodana je datoteci „android/build.gradle“. Ovaj classpath je osigurao da su *Firebase* usluge ispravno integrirane u postavke projekta za Android.

```
dependencies {
    implementation platform('com.google.firebase:firebase-bom:32.1.0')
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
}
```

Slika 7 Ovisnosti za Firebase (App razina)

```
8       dependencies {
9           classpath 'com.google.gms:google-services:4.3.15'
10          classpath 'com.android.tools.build:gradle:7.3.0'
11          classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
12      }
13  }
14
```

Slika 8 Ovisnosti za Firebase (Android razina)

Preuzeto je nekoliko paketa kako bi aplikacija mogla koristiti *Firebase* servise. Potrebni paketi bili su "cloud_firestore" za povezivanje *Firestore* baze podataka, "firebase_auth" za mogućnosti autentifikacije, "firebase_core" za osnovne značajke *Firebase-a* i "shared_preferences" za lokalno pohranjivanje korisničkih postavki. Ovi su paketi dali aplikaciji pristup potrebnim *Firebase* uslugama, omogućujući nesmetano upravljanje podacima i komunikaciju između aplikacije i *Firebase-a*.

4. Analiza programskog koda

4.1. Datoteka „main.dart“

Datoteka „main.dart“ djeluje kao ulazna točka aplikacije i zadužena je za njezino pokretanje, održavanje ruta i odabir prvog zaslona koji će se prikazati ovisno o stanju autentifikacije korisnika. Brojne konstante i ovisnosti se uvoze na početku datoteke. Oni se sastoje od datoteke „helper_functions.dart“, stranice za prijavu, početne stranice i datoteke „constants.dart“, koja pruža ključne podatke za inicijalizaciju *Firebase-a*, uključujući API ključ, ID aplikacije, ID pošiljatelja poruka i ID projekta.

```
1  import 'package:chat_app/helper/helper_functions.dart';
2  import 'package:chat_app/pages/auth/login_page.dart';
3  import 'package:chat_app/pages/home_page.dart';
4  import 'package:chat_app/shared/constants.dart';
5  import 'package:flutter/foundation.dart';
6  import 'package:flutter/material.dart';
7  import 'package:firebase_core/firebase_core.dart';
8
```

Slika 9 Uvoz konstanti i ovisnosti u main.dart datoteku

Asinkrona funkcija „main()“ služi kao početna točka aplikacije. Osigurava da su *Flutter* vezivanja pravilno inicijalizirana pomoću „WidgetsFlutterBinding.ensureInitialized()“. Ovisno o platformi (web ili non-web), *Firebase* se inicijalizira u skladu s tim. „Firebase.initializeApp()“ poziva se s „FirebaseOptions“, što uključuje potrebne vjerodajnice kao što su API ključ, ID aplikacije, ID pošiljatelja poruka i ID projekta, ako se aplikacija izvodi na webu (kIsWeb). Inače se prilikom pozivanja „Firebase.initializeApp()“ ne prosljeđuju parametri. Glavni widget aplikacije predstavlja klasa „MyApp“, koja proširuje „StatefulWidget“. Kontrolira status autentifikacije korisnika i pohranjuje stanje (_MyAppState). Stanje autentifikacije korisnika prati boolean [9]varijabla „_isSignedIn“ unutar klase „_MyAppState“. Funkcija „getUserLoggedInStatus()“ koristi se u metodi „initState()“ za dobivanje statusa prijave iz klase „HelperFunctions“. Ako vrijednost nije null kada je primljena,

[9] Podatkovni tip koji ima dvije moguće vrijednosti: istina (engl. *true*) ili laž (engl. *false*)

„setState()“ se koristi za promjenu stanja „_isSignedIn“. UI aplikacije izgrađen je pomoću funkcije „build()“ klase „_MyAppState“. Root widget, poznat kao „MaterialApp“, uspostavlja opće postavke i temu za aplikaciju. Primarna boja se dohvaća iz klase „Constants“ prilikom postavljanja teme pomoću „ThemeData“. To jamči kontinuitet u vizualnom izgledu aplikacije. Da bi se sakrio debug banner, svojstvo „debugShowCheckedModeBanner“ postavljeno je na *false*[10]. Početno svojstvo „MaterialApp-a“ uvjetno je postavljeno na temelju statusa „_isSignedIn“. Ako je „_isSignedIn“ *true*, korisnik je prijavljen, a svojstvo „home“ postavljeno je na widget „HomePage“. Ako korisnik nije prijavljen („_isSignedIn“ je *false*), početno svojstvo postavljeno je na widget „LoginPage“. Instanca „MyApp-a“ prosljeđuje se funkciji „runApp()“ kao parametar, čime se pokreće aplikacija.

[10] Logička vrijednost (boolean) koja označava netočnost.

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  if (kIsWeb) {
    await Firebase.initializeApp(
        options: FirebaseOptions(
            apiKey: Constants.apiKey,
            appId: Constants.appId,
            messagingSenderId: Constants.messagingSenderId,
            projectId: Constants.projectId));
  } else {
    await Firebase.initializeApp();
  }
  runApp(const MyApp());
}

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  bool _isSignedIn = false;

  @override
  void initState() {
    super.initState();
    getUserLoggedInStatus();
  }

  getUserLoggedInStatus() async {
    await HelperFunctions.getUserLoggedInStatus().then((value) {
      if (value != null) {
        setState(() {
          _isSignedIn = value;
        });
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primaryColor: Constants().primaryColor
      ), // ThemeData
      debugShowCheckedModeBanner: false,
      home: _isSignedIn ? const HomePage() : const LoginPage(),
    ); // MaterialApp
  }
}
```

Slika 10 Sadržaj datoteke main.dart

4.2. Folder „helper“

4.2.1. Datoteka „helper_functions.dart“

Datoteka „helper_functions.dart“ sadrži zbirku statičkih metoda koje pružaju funkcionalnost za upravljanje korisničkim podacima i njihovo održavanje pomoću paketa „SharedPreferences“. Ove pomoćne funkcije igraju ključnu ulogu u osiguravanju besprijekornog i personaliziranog korisničkog iskustva unutar aplikacije. Paket „shared_preferences“ uvozi se na početku programa kako bi se koristile značajke „SharedPreferences“. Aplikacija može pohraniti i dohvatiti korisničke podatke kroz nekoliko sesija aplikacije zahvaljujući jednostavnoj i učinkovitoj uporabi ovog paketa.

```
1   import 'package:shared_preferences/shared_preferences.dart';
```

Slika 11 Uvođenje paketa "SharedPreferences"

Na početku funkcije „helper_functions“ definirano je nekoliko statičkih nizova koji služe kao ključevi korisničkih podataka. „UserNameKey“, „UserEmailKey“ i „UserLoggedInKey“ neki su od ovih ključeva. Određeni korisnički podaci bit će dohvaćeni i pohranjeni u „SharedPreferences“ pomoću ovih ključeva.

```
4   //Kljucevi
5   static String userLoggedInKey = "LOGGEDINKEY";
6   static String userNameKey = "USERNAMEKEY";
7   static String userEmailKey = "USEREMAILKEY";
```

Slika 12 Ključevi za dohvat i spremanje podataka preko SharedPreferences paketa

Datoteka „helper_functions.dart“ pruža metode za spremanje korisničkih podataka u „SharedPreferences“. Korisnički status prijave, ime i adresa e-pošte spremaju se putem funkcija „saveUserNameSF(„) i „saveUserEmailSF()“. Svaka metoda koristi „SharedPreferences“ za dobivanje instance. Podaci se naknadno pohranjuju pomoću odgovarajuće funkcije skupa (setBool(), setString()).

Za dohvaćanje korisničkih podataka iz „SharedPreferences“, datoteka „helper_functions.dart“ nudi odgovarajuće metode. „GetUserEmailSF()“ dobiva adresu e-pošte korisnika kao *Future* String, „getUserNameSF()“ dobiva korisničko ime kao *Future* String, a „getUserLoggedInStatus()“ dobiva korisnikov status prijave kao *Future* boolean. Pohranjeni podaci se dohvaćaju pomoću ekvivalentne get[11] metode (getBool(), getString()). Sve asinkrone metode u datoteci „helper_functions.dart“ vraćaju *Future*. To omogućuje programu da izvršava druge zadatke dok čeka da se procesi dohvaćanja podataka ili pohrane završe. Metode mogu bez napora komunicirati sa „SharedPreferences“ bez ometanja glavne funkcije korištenjem ključnih riječi *async*[12] i *await*[13].

```dart
class HelperFunctions {
  //Kljucevi
  static String userLoggedInKey = "LOGGEDINKEY";
  static String userNameKey = "USERNAMEKEY";
  static String userEmailKey = "USEREMAILKEY";

  static Future<bool> saveUserLoggedInStatus(bool isUserLoggedIn) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setBool(userLoggedInKey, isUserLoggedIn);
  }

  static Future<bool> saveUserNameSF(String userName) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userNameKey, userName);
  }

  static Future<bool> saveUserEmailSF(String userEmail) async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return await sf.setString(userEmailKey, userEmail);
  }

  static Future<bool?> getUserLoggedInStatus() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getBool(userLoggedInKey);
  }

  static Future<String?> getUserEmailSF() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getString(userEmailKey);
  }

  static Future<String?> getUserNameSF() async {
    SharedPreferences sf = await SharedPreferences.getInstance();
    return sf.getString(userNameKey);
  }
}
```

Slika 13 Sadržaj datoteke „helper_functions.dart“

[11] Metode u programiranju koje se koriste za pristupanje i dobivanje vrijednosti privatnih atributa (članova) klase.
[12] Označava asinkronu funkciju.
[13] Koristi se unutar asinkronih funkcija za čekanje da se završi neka asinkrona operacija.

4.3. Folder „service“

4.3.1. Datoteka „auth_service.dart“

Klasa „AuthService“ nudi načine za prijavu, registraciju i odjavu korisnika. Metoda „signInWithEmailAndPassword()“ koju nudi „FirebaseAuth“ koristi se za pokušaj autentifikacije korisnika kada metoda „loginUser()“ primi korisnikovu e-poštu i lozinku za ulaz. Metoda vraća autentificirani korisnički objekt ako je prijava uspješna. Kako bi primila korisničke podatke iz *Firestore* baze podataka, poziva se metoda „saveUserData()“ iz „DatabaseService“. Korisnici mogu registrirati nove račune pomoću metode „registerUser()“. Kao argumente uzima puno ime korisnika, adresu e-pošte i lozinku. Aplikacija generira novi korisnički račun koristeći isporučenu adresu e-pošte i lozinku pomoću „FirebaseAuth-ove“ metode „createUserWithEmailAndPassword()“. Ako je proces stvaranja računa uspješan, funkcija koristi metodu „saveUserData()“ usluge „DatabaseService“ za spremanje podataka o korisniku u *Firestore* bazu podataka. Postupak odjave za korisnike je olakšan metodom „signOut()“. Koristeći metodu „saveUserLoggedInStatus()“ iz klase „HelperFunctions“, mijenja se stanje prijavljenog korisnika iz *true* u *false*. Pomoću metoda „saveUserNameSF()“ i „saveUserEmailSF()“ također se brišu ime i e-pošta korisnika. Za odjavu korisnika konačno se koristi „FirebaseAuth-ova„ metoda „signOut()“. Sve pogreške koje se dogode tijekom postupka provjere autentičnosti bilježi „FirebaseAuthException“, koji vraća poruku o pogrešci. To jamči da se svi neuspjeli pokušaji prijave ili registracije ispravno obrađuju i prikazuju korisniku.

```dart
class AuthService {
  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;

  //Login
  Future loginUser(String email, String password) async {
    try {
      User user = (await firebaseAuth.signInWithEmailAndPassword(
              email: email, password: password))
          .user!;

      if (user != null) {
        //Poziv na database
        return true;
      }
    } on FirebaseAuthException catch (e) {
      return e.message;
    }
  }

//Registracija
  Future registerUser(String fullName, String email, String password) async {
    try {
      User user = (await firebaseAuth.createUserWithEmailAndPassword(
              email: email, password: password))
          .user!;

      if (user != null) {
        //Poziv na database
        await DatabaseService(uid: user.uid).saveUserData(fullName, email);
        return true;
      }
    } on FirebaseAuthException catch (e) {
      return e.message;
    }
  }

  //Odjava
  Future signOut() async {
    try {
      await HelperFunctions.saveUserLoggedInStatus(false);
      await HelperFunctions.saveUserNameSF("");
      await HelperFunctions.saveUserEmailSF("");
      await firebaseAuth.signOut();
    } catch (e) {
      return null;
    }
  }
}
```

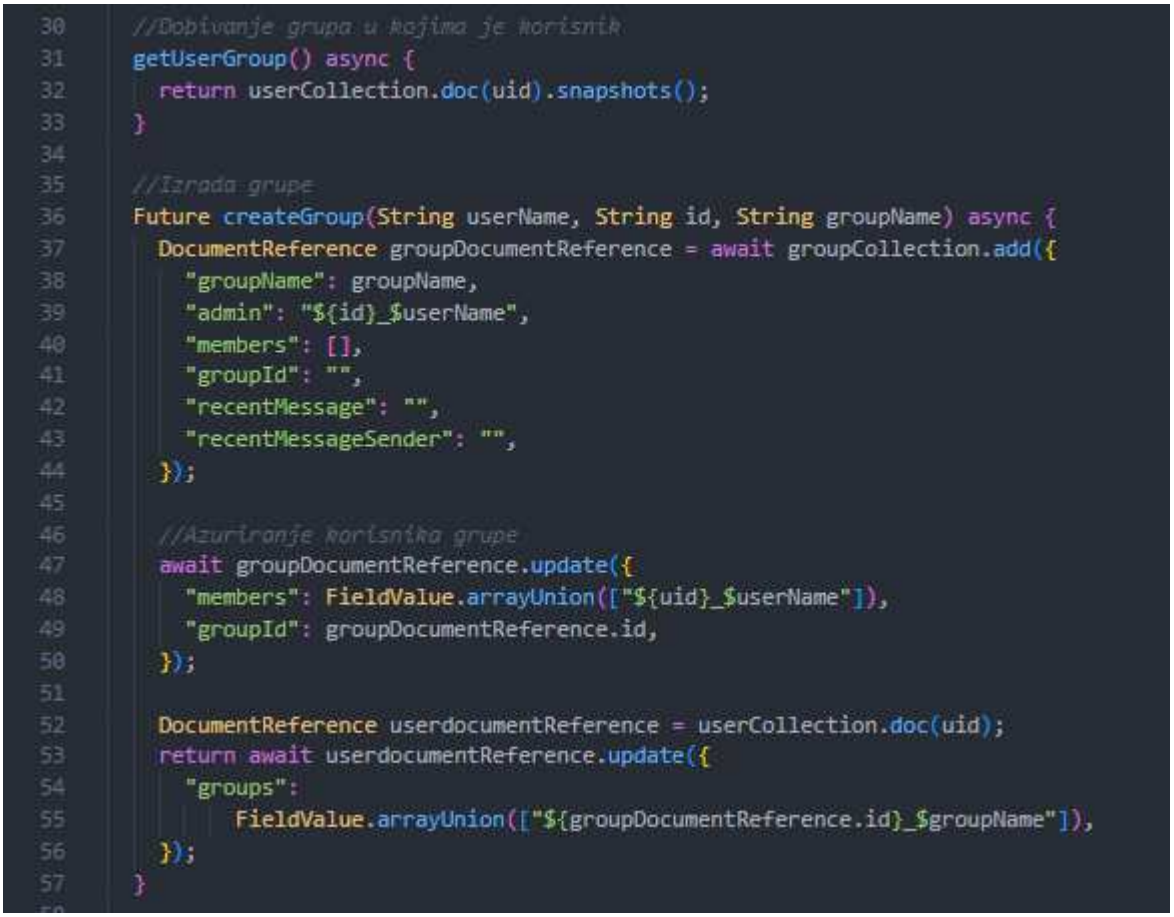
Slika 14 Sadržaj datoteke "auth_service.dart

4.3.2. Datoteka „database_service.dart“

Klasa „DatabaseService“ upravlja operacijama baze podataka. Nudi načine za stvaranje grupa, pronalaženje informacija o grupi, upravljanje članstvom u grupi, traženje grupa, spremanje i dohvaćanje korisničkih podataka i slanje poruka. Podaci o korisniku, uključujući puno ime korisnika, adresu e-pošte, grupe i UID, spremaju se pomoću metode „saveUserData()“ i čuvaju se u zbirci "users“. Postavljaju se potrebna polja i stvara se novi dokument s UID-om kao ID-om dokumenta. Na temelju dostavljene e-pošte, metoda „getUserData()“ izvlači korisničke podatke iz *Firestore* baze podataka. Upituje se zbirka "users", a dokumenti se filtriraju prema tome odgovara li vrijednost u polju "email" danoj e-pošti. Rezultat metode je QuerySnapshot koji sadrži podudarne dokumente. Nova grupa se može stvoriti pomoću metode „createGroup()“. Stvara novi dokument s poljima kao što su „groupName“, „admin“, „members“, „groupId“, „recentMessage“ i „recentMessageSender“ i dodaje ga u kolekciju "groups". UID i naziv grupe koriste se kao identifikatori u metodi za ažuriranje korisničkog dokumenta i dodavanje grupe u polje "groups".

```
13  //Spremanje podataka o korsniku
14  Future saveUserData(String fullName, String email) async {
15    return await userCollection.doc(uid).set({
16      "fullName": fullName,
17      "email": email,
18      "groups": [],
19      "uid": uid,
20    });
21  }
22
23  //Dobivanje podataka o korisniku
24  Future getUserData(String email) async {
25    QuerySnapshot snapshot =
26        await userCollection.where("email", isEqualTo: email).get();
27    return snapshot;
28  }
29
```

Slika 15 Metode "saveUserData()" i "getUserData()"



```
//Dobivanje grupa u kojima je korisnik
getUserGroup() async {
  return userCollection.doc(uid).snapshots();
}

//Izrada grupe
Future createGroup(String userName, String id, String groupName) async {
  DocumentReference groupDocumentReference = await groupCollection.add({
    "groupName": groupName,
    "admin": "${id}_$userName",
    "members": [],
    "groupId": "",
    "recentMessage": "",
    "recentMessageSender": "",
  });

  //Azuriranje korisnika grupe
  await groupDocumentReference.update({
    "members": FieldValue.arrayUnion(["${uid}_$userName"]),
    "groupId": groupDocumentReference.id,
  });

  DocumentReference userdocumentReference = userCollection.doc(uid);
  return await userdocumentReference.update({
    "groups":
        FieldValue.arrayUnion(["${groupDocumentReference.id}_$groupName"]),
  });
}
```

Slika 16 Metode "createGroup()" i "getUserGroup()"

Podzbirci "messages" odabranog grupnog dokumenta pristupa se metodom „getChats()“ za dohvaćanje *chat* poruka iz određene grupe. Poruke se vraćaju kao *Stream* i sortirane po vremenu. Pomoću metode „toggleGroupJoin()“ korisnici mogu ući u grupu ili izaći iz nje. Skeniranjem korisničkog polja "groups" u potrazi za kombinacijom „groupId“ i „groupName“, utvrđuje se je li korisnik već član grupe. Metoda briše grupu iz korisničkog polja "groups" i korisnika iz polja "members" grupe ako je korisnik već član. S druge strane, ako korisnik nije član, postupak dodaje korisnika u polje "members" grupe, a grupu u polje "groups" korisnika.

```
//Dobivanje chatova
getChats(String groupId) async {
  return groupCollection
      .doc(groupId)
      .collection("messages")
      .orderBy("time")
      .snapshots();
}
```

Slika 17 Metoda "getChats()"

```dart
// Ulaz / izlaz
Future toggleGroupJoin(
    String groupId, String userName, String groupName) async {
  DocumentReference uRef = userCollection.doc(uid);
  DocumentReference gRef = groupCollection.doc(groupId);

  DocumentSnapshot dSnap = await uRef.get();
  List<dynamic> groups = await dSnap['groups'];

  if (groups.contains("${groupId}_$groupName")) {
    //Izlazak iz grupe
    await uRef.update({
      "groups": FieldValue.arrayRemove(["${groupId}_$groupName"]),
    });
    await gRef.update({
      "members": FieldValue.arrayRemove(["${uid}_$userName"]),
    });
  } else {
    //Ulazak u grupu
    await uRef.update({
      "groups": FieldValue.arrayUnion(["${groupId}_$groupName"]),
    });
    await gRef.update({
      "members": FieldValue.arrayUnion(["${uid}_$userName"]),
    });
  }
}

```

Slika 18 Metoda "toggleGroupJoin()“

Korisnici mogu tražiti grupe pomoću naziva grupe pomoću funkcije „searchGroupsByName()“. Pokreće se upit o zbirci "groups" i filtriraju se svi dokumenti u kojima je naziv grupe prisutan u stupcu "groupName". Metoda „sendMessage()“ za upravljanje porukama šalje *chat* poruku određenoj grupi. Podzbirka "messages" odabranog grupnog dokumenta prima podatke o poruci kao novi dokument. Osim toga, dodaje sadržaj poruke i podatke o pošiljatelju u odjeljke "recentMessage" i "recentMessageSender" grupnog dokumenta.

```dart
//Pretrazivanje grupa po imenu
searchGroupsByName(String groupName) {
  return groupCollection.where("groupName", isEqualTo: groupName).get();
}

Future<bool> isUserInGroup(
    String groupName, String groupId, String userName) async {
  DocumentReference uRef = userCollection.doc(uid);
  DocumentSnapshot dSnap = await uRef.get();

  List<dynamic> groups = await dSnap['groups'];
  if (groups.contains("${groupId}_$groupName")) {
    return true;
  } else {
    return false;
  }
}
```

Slika 19 Metoda "searchGroupsByName()"

```
125   //Slanje poruka
126   sendMessage(String groupId, Map<String, dynamic> chatMessageData) async {
127     groupCollection.doc(groupId).collection("messages").add(chatMessageData);
128     groupCollection.doc(groupId).update({
129       "recentMessage": chatMessageData["message"],
130       "recentMessageSender": chatMessageData["sender"],
131     });
132   }
```

Slika 20 Metoda "sendMessage()"

4.4. Folder „shared“

4.4.1. Datoteka „constants.dart“

Datoteka „constants.dart“, sadrži vitalne konstante koje se koriste u cijeloj aplikaciji. Ove konstante sadrže ID-ove projekta, kodove boja i druge informacije o konfiguraciji. Klasa „Constants“ osigurava da su ove varijable dostupne i dosljedno korištene u cijeloj aplikaciji centralizirajući ih u jednoj datoteci. Prilikom povezivanja s raznim uslugama ili API-ima unutar programa, ove se vrijednosti koriste za provjeru autentičnosti i identitet. Na primjer, „projectId“ se koristi za identifikaciju ID-a projekta povezanog s aplikacijom, dok su „appId“ i „apiKey“ potrebni za *Firebase* autentifikaciju i pristup *Firestore* bazi podataka. Konstante boja također su opisane u klasi i koriste se za održavanje sheme boja aplikacije konstantnom. Kapsuliranjem ovih konstanti u zasebnu datoteku, aplikacija ima koristi od poboljšane organizacije koda, mogućnosti ponovne upotrebe i lakšeg održavanja.

```
2
3   class Constants {
4     static String appId = "1:139432703985:web:c14c2a8d11eb837bd57650";
5     static String apiKey = "AIzaSyBF7RUj0FThBmONoD1S5Yapw2oYbWepTVs";
6     static String messagingSenderId = "139432703985";
7     static String projectId = "chatappflutter-4b438";
8     final primaryColor = const Color(0xFFFFD369);
9   /*    final secondaryColor = const Color(0xFF393E46);
10    final primaryWhite = const Color(0xFFEEEEEE);
11    final primaryBlack = const Color(0xFF222831); */
12  }
```

Slika 21 Sadržaj datoteke "constants.dart"

4.5. Folder „widgets“

4.5.1. Datoteka „widgets.dart“

Ova datoteka sadrži konstantne varijable i funkcije koje mogu koristiti različiti widgeti i zasloni u cijeloj aplikaciji. Instanca klase „InputDecoration“ predstavljena je konstantom „textInputDecoration“. Za polja za unos aplikacije kao što su tekstualna polja, određuje vizualni dizajn. Svojstva poput „labelStyle“, „focusedBorder“, „enabledBorder“ i „errorBorder“ uključena su u konstantu. Funkcija „nextScreen“ je navigacijska pomoć koja pojednostavljuje promjene zaslona unutar aplikacije. Parametar stranice označava konačnu stranicu ili ekran do kojeg treba doći, dok parametar konteksta označava trenutni kontekst aplikacije. Ova funkcija koristi klasu „Navigator“ za prijelaz na sljedeći zaslon guranjem ponuđene stranice na navigacijski stog. Dok funkcija „nextScreen“ gura novi zaslon na stog, funkcija „nextScreenReplace“ provodi zamjensku navigaciju. Ova funkcija zamjenjuje isporučenu stranicu ili zaslon za trenutni zaslon. Ona je od pomoći kada se želite pomaknuti na novi zaslon dok brišete stari zaslon iz navigacijskog niza. Funkcija „showSnackBar“ korisnicima daje jednostavnu metodu za pregled trenutnih upozorenja na *snack* baru. Kontekst, boja i poruka tri su čimbenika koja razmatra. Argument konteksta označava trenutni kontekst aplikacije, parametar boje uspostavlja boju pozadine *snack* trake, a parametar poruke označava tekst koji će biti prikazan na *snack* traci. Ova funkcija prikazuje *snack* traku sa željenom bojom, porukom i trajanjem pomoću klase „ScaffoldMessenger“. Dodatno, postoji akcijski gumb "OK".

```dart
const textInputDecoration = InputDecoration(
    labelStyle:
        TextStyle(color: Color(0xFFEEEEEE), fontWeight: FontWeight.w300),
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFFFFD369), width: 2),
    ), // OutlineInputBorder
    enabledBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFFFFD369), width: 2),
    ), // OutlineInputBorder
    errorBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Color(0xFFFFD369), width: 2),
    )); // OutlineInputBorder // InputDecoration

void nextScreen(context, page) {
  Navigator.push(context, MaterialPageRoute(builder: (context) => page));
}

void nextScreenReplace(context, page) {
  Navigator.pushReplacement(
      context, MaterialPageRoute(builder: (context) => page));
}

void showSnackBar(context, color, message) {
  ScaffoldMessenger.of(context).showSnackBar(SnackBar(
    backgroundColor: color,
    content: Text(message, style: TextStyle(fontSize: 14)),
    duration: const Duration(seconds: 3),
    action: SnackBarAction(
      label: "OK",
      onPressed: () {},
      textColor: Colors.white,
    ), // SnackBarAction
  )); // SnackBar
}
```

Slika 22 Sadržaj datoteke "widgets.dart"

4.5.2. Datoteka „groupItem.dart“

Widget GroupItem je zadužen za predstavljanje relevantnih detalja o grupi, kao što su naziv grupe, korisničko ime admina i opcije za ulazak u grupu. Tijekom inicijalizacije prihvaća potrebne unose korisničko ime, ID grupe i ime grupe. Ovi parametri nude informacije potrebne za ispravno popunjavanje grupne stavke. Widget „GroupItem“ koristi „GestureDetector[14]“ za detekciju dodira korisnika. „OnTap“ povratni poziv aktivira se kada korisnik dodirne grupnu stavku i usmjerava ga na odgovarajuću „ChatPage“. Widget „Container“ koji obuhvaća „ListTile“ čini vizualni prikaz widgeta. „ListTile[15]“ je primarni spremnik sadržaja za predstavljanje informacija o grupi. Postoji widget „CircleAvatar“ u vodećem dijelu „ListTile-a“. Profilna slika „CircleAvatara“ je početno slovo naziva grupe. Boja pozadine postavljena je na primarnu boju navedenu u temi aplikacije pomoću atributa „BackgroundColor“. Atribut radijusa „CircleAvatara“ postavlja njegovu veličinu. Naziv grupe prikazan je kao tekstualni widget u odjeljku naslova „ListTile-a“. Stil teksta određen je atributima „fontWeight“, „fontSize“ i „color“. Odjeljak podnaslova „ListTile“ prikazuje opisnu poruku koristeći ime korisnika. Obavještava korisnika da se može pridružiti *chat-u* kao navedeni korisnik. Slično odjeljku naslova, odjeljak titlova koristi widget „Text“ s odgovarajućim svojstvima stila.

[14] Widget u Flutteru koji omogućava detekciju različitih korisničkih gesti na određenom dijelu korisničkog sučelja.
[15] Flutter widget, koristi za prikazivanje jednog reda ili stavke unutar liste ili popisa.

```dart
class GroupItem extends StatefulWidget {
  final String userName;
  final String groupId;
  final String groupName;
  const GroupItem(
      {Key? key,
      required this.userName,
      required this.groupId,
      required this.groupName})
      : super(key: key);

  @override
  State<GroupItem> createState() => _GroupItemState();
}

class _GroupItemState extends State<GroupItem> {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        nextScreen(
            context,
            ChatPage(
              groupId: widget.groupId,
              groupName: widget.groupName,
              userName: widget.userName,
            )); // ChatPage
      },
      child: Container(
        padding: const EdgeInsets.symmetric(horizontal: 5, vertical: 10),
        child: ListTile(
          leading: CircleAvatar(
            backgroundColor: Theme.of(context).primaryColor,
            radius: 30,
            child: Text(
              widget.groupName.substring(0, 1).toUpperCase(),
              textAlign: TextAlign.center,
              style: const TextStyle(
                  color: Color(0xFF393E46), fontWeight: FontWeight.w600), // TextStyle
            ), // Text
          ), // CircleAvatar
          title: Text(
            widget.groupName,
            style: const TextStyle(
              fontWeight: FontWeight.w500,
              fontSize: 18,
              color: Color(0xFFEEEEEE),
            ), // TextStyle
          ), // Text
          subtitle: Text(
            "Join the chat as ${widget.userName}",
            style: const TextStyle(fontSize: 14, color: Color(0xFFEEEEEE)),
          ), // Text
        ), // ListTile
      ), // Container
    ); // GestureDetector
  }
}
```

Slika 23 Sadržaj datoteke "group_item.dart"

4.5.3. Datoteka „message_tile.dart“

„MessageTile“ widget predstavlja jednu *chat* poruku unutar sučelja za *chat*. Ovaj widget je zadužen za prikaz sadržaja poruke, pojedinosti o pošiljatelju te vizualno odvajanje poruka koje šalje korisnik od onih koje šalju drugi. Tijekom inicijalizacije potrebni su parametri „message“, „sender“ i „sentByMe“. Ovi parametri daju informacije potrebne za ispunjavanje pločice poruke ispravnim sadržajem. Vizualni prikaz widgeta „MessageTile“ sastoji se od spremnika koji okružuje drugi spremnik. Ovaj ugniježđeni spremnik služi kao glavni spremnik sadržaja za prikaz poruke i informacija o pošiljatelju. Ovisno o tome je li pločicu poruke poslao korisnik ili neki drugi sudionik *chat-a*, vanjski spremnik je zadužen za njeno postavljanje i poravnavanje. Oblačić poruke predstavljen je unutarnjim spremnikom. Ovisno o tome je li poruku poslao korisnik ili ne, ona koristi atribut „margin[16]“ za dodatni razmak na lijevoj ili desnoj strani oblačića. Unutarnjim razmakom oblačića tada se upravlja pomoću atributa „padding[17]“. To je li poruku poslao korisnik ili netko drugi utječe na to kako će spremnik biti ukrašen. Osim toga, parametar „borderRadius“ koristi se za zaokruživanje kutova oblačića poruke. Ime pošiljatelja i tekst poruke prikazani su pomoću dva tekstualna widgeta u odjeljku sadržaja unutarnjeg spremnika. Oblačić poruke je centriran, a ime pošiljatelja ispisano je velikim slovima. Vanjski izgled imena pošiljatelja definiran je stilskim elementima kao što su veličina fonta, težina fonta, boja i razmak između slova.

[16] Razmak između elementa (widgeta) i okolnih elemenata.
[17] Razmak unutar elementa (widgeta), između sadržaja tog elementa i njegove granice.

```dart
class MessageTile extends StatefulWidget {
  final String message;
  final String sender;
  final bool sentByMe;
  const MessageTile(
      {Key? key,
      required this.message,
      required this.sender,
      required this.sentByMe})
      : super(key: key);

  @override
  State<MessageTile> createState() => _MessageTileState();
}

class _MessageTileState extends State<MessageTile> {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.only(
          top: 4,
          bottom: 4,
          left: widget.sentByMe ? 0 : 14,
          right: widget.sentByMe ? 14 : 0), // EdgeInsets.only
      alignment: widget.sentByMe ? Alignment.centerRight : Alignment.centerLeft,
      child: Container(
        margin: widget.sentByMe
            ? const EdgeInsets.only(left: 30)
            : const EdgeInsets.only(right: 30),
        padding:
            const EdgeInsets.only(top: 15, bottom: 15, left: 22, right: 22),
        decoration: BoxDecoration(
            color:
                widget.sentByMe ? Theme.of(context).primaryColor : Colors.grey,
            borderRadius: widget.sentByMe
                ? const BorderRadius.only(
                    topLeft: Radius.circular(20),
                    topRight: Radius.circular(20),
                    bottomLeft: Radius.circular(20)) // BorderRadius.only
                : const BorderRadius.only(
                    topLeft: Radius.circular(20),
                    topRight: Radius.circular(20),
                    bottomRight: Radius.circular(20))), // BorderRadius.only // BoxDecoration
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              widget.sender.toUpperCase(),
              textAlign: TextAlign.center,
              style: const TextStyle(
                fontSize: 13,
                fontWeight: FontWeight.w700,
                color: Color(0xFF393E46),
                letterSpacing: -0.5,
              ), // TextStyle
            ), // Text
            const SizedBox(
              height: 8,
            ), // SizedBox
            Text(
              widget.message,
              textAlign: TextAlign.center,
              style: const TextStyle(fontSize: 16, color: Color(0xFF393E46)),
            ), // Text
          ],
        ), // Column
      ), // Container
    ); // Container
  }
}
```

Slika 24 Sadržaj datoteke "message_tile.dart"

4.6. Folder „pages“

Zbog znatne količine programskog koda koji je uključen u implementaciju svih stranica aplikacije, uključivanje slika koda u ovaj dio završnog rada bilo bi nepraktično. Međutim, pružat će se detaljno objašnjenje ključnih komponenti i njihovih funkcija.

4.6.1. Datoteka „login_page.dart“

Stranica za prijavu je odgovorna za autentifikaciju korisnika i dopuštanje pristupa funkcionalnostima aplikacije. Korisnici mogu unijeti svoje vjerodajnice i sigurno se prijaviti pomoću korisničkog sučelja. *Flutter* widgeti i *Firebase* API-ji koriste se za implementaciju dizajna i funkcionalnosti stranice za prijavu. Korisnike dočekuje vizualno privlačno i jednostavno sučelje kada uđu na stranicu za prijavu. Za izradu komponenti korisničkog sučelja koriste se principi *Flutter-ovog* materijalnog dizajna, što rezultira jedinstvenim i ugodnim korisničkim sučeljem. Na stranici su prisutni logotip aplikacije i sažeta poruka koja poziva korisnike da se prijave i sudjeluju u *chat-u*. Kako bi se postigla potrebna funkcionalnost, stranica za prijavu koristi brojne *Flutter* widgete i pomoćne funkcije. Korištenjem widgeta „TextFormField“, koji nude unos teksta s podesivim stilom i provjerom valjanosti, razvijena su korisnička polja za unos e-pošte i lozinke. Oznake, ikone i stilovi obruba za ova polja za unos definirani su konstantom „textInputDecoration[18]“ koja se nalazi u datoteci „widgets.dart“. Regularni izrazi koji su definirani unutar funkcija valjanosti widgeta „TextFormField“ koriste se za provjeru korisničkog unosa. Validator[19] lozinke provodi posebne zahtjeve zaporke, kao što su uključivanje velikih slova, znamenki i minimalne duljine od osam znakova, dok validator e-pošte provjerava je li unesena vrijednost legitimna adresa e-pošte. Korisniku se daju odgovarajuće poruke o pogrešci ako unos ne prođe provjeru valjanosti. Kada se klikne, gumb „Prijava“, koji je dizajniran s primarnom bojom aplikacije, pokreće se proces prijave. Funkcija

[18] Klasa u Flutteru koja se koristi za definiranje izgleda i ponašanja dekoracije polja unosa (input field).
[19] Funkcija koja omogućava provjeru unesenih vrijednosti prije nego što se podaci pošalju na daljnju obradu ili spremanje.

prijave, koja provodi proceduru provjere autentičnosti, poziva se povratnim pozivom „onPressed“. Funkcija prijave provjerava je li unos obrasca ispravan, postavlja stanje učitavanja na istinito (_isLoading), zatim poziva funkciju „loginUser“ usluge „AuthService“ za provjeru autentičnosti korisnika pomoću dostavljene e-pošte i lozinke. Klasa „DatabaseService“ koristi se za dohvaćanje korisničkih podataka iz *Firestore* baze podataka ako je autentifikacija uspješna. Klasa „HelperFunctions“ koristi zajedničke postavke za trajno pohranjivanje parova ključ-vrijednost i lokalno sprema status prijavljenog korisnika, e-poštu i ime. Na kraju, koristeći pomoćnu metodu „nextScreenReplace“, korisnik se usmjerava na početnu stranicu. Ako provjera autentičnosti ne uspije, pomoću pomoćne funkcije „showSnackBar“ prikazuje se poruka o pogrešci s uzrokom greške. Status učitavanja mijenja se natrag u *false*, što korisniku omogućuje ponovni pokušaj prijave ili poduzimanje potrebnih radnji.

4.6.2. Datoteka „register_page.dart“

Na stranici za registraciju korisnici mogu registrirati nove račune. Korisnici mogu unijeti svoje osobne podatke i sigurno se registrirati zahvaljujući prilagođenom korisničkom sučelju. *Flutter* widgeti koriste se za implementaciju stranice za registraciju, koja koristi *Firebase* API-je za autentifikaciju korisnika i pohranu podataka. Kada korisnici uđu na stranicu dočekuje atraktivno i jednostavno sučelje. Funkcionalnost stranice za registraciju implementirana je pomoću raznih *Flutter* widgeta i pomoćnih funkcija. Korisnička polja za unos punog imena, e-pošte i lozinke stvaraju se pomoću widgeta „TextFormField“, koji nude unos teksta s ugrađenim postavkama stila i provjere valjanosti. Oznake, ikone i stilovi obruba za ova polja za unos definirani su konstantom „textInputDecoration“ koja se nalazi u datoteci „widgets.dart“. Regularni izrazi koji su definirani unutar funkcija valjanosti widgeta „TextFormField“ koriste se za provjeru korisničkog unosa. Dok polje e-pošte provjerava je li isporučena vrijednost legitimna adresa e-pošte, polje za potpuno ime zahtijeva vrijednost koja nije prazna. Posebna ograničenja lozinke, poput upotrebe velikih slova, znamenki i minimalne duljine od osam znakova, nametnuta su poljem lozinke. Ako bilo koji unos ne prođe provjeru valjanosti, korisnik će vidjeti

odgovarajuće poruke o pogrešci. Primarna boja aplikacije koristi se za ukrašavanje gumba za registraciju. Gumb aktivira funkciju registracije, koja pokreće postupak registracije, kada se pritisne. Funkcija „register“ provjerava unos obrasca i postavlja stanje učitavanja (_isLoading) na *true*[20] ako je provjera valjanosti uspješna. Kako bi se stvorio novi korisnički račun koji koristi usluge provjere autentičnosti *Firebase*, tada se koristi funkcija „registerUser“ usluge“ AuthService“, dajući puno ime, adresu e-pošte i lozinku. Korištenjem klase „HelperFunctions“, koja iskorištava zajedničke postavke za trajno pohranjivanje parova ključ-vrijednost, podaci o korisniku spremaju se lokalno ako je postupak registracije uspješan. Puno ime, adresa e-pošte i status prijavljenog korisnika spremaju se za kasniju upotrebu. „NextScreenReplace“ pomoćna funkcija zamjenjuje stranicu za registraciju prije navigacije korisnika na početnu stranicu. Ako je pokušaj registracije neuspješan, prikazuje se poruka o pogrešci s objašnjenjem greške. Kada se status učitavanja vrati na *false*, korisnik može ponovno pokušati s procedurom registracije ili poduzeti potrebne radnje.

4.6.3. Datoteka „home_page.dart“

Početna stranica aplikacije za *chat* služi kao središnja točka, dajući korisnicima personalizirano i jednostavno sučelje za upravljanje svojim grupama i istraživanje platforme. Korisnici se nakon uspješne prijave preusmjeravaju na glavnu stranicu, gdje mogu pristupiti nizu usluga i funkcionalnosti. Na vrhu početne stranice nalazi se traka aplikacije s nazivom ili logotipom aplikacije i ikonom za pretraživanje. Unutar aplikacije korisnici mogu upotrijebiti ikonu pretraživanja za traženje određenih grupa. „Body“ početne stranice prikazuje primarni sadržaj stranice, koji je prvenstveno popis korisničkih grupa. Element "GroupItem" predstavlja svaku grupu. Ako se korisnik još nije pridružio nijednoj grupi, na glavnoj stranici prikazuje se drugi odjeljak u kojem je korisnik obaviješten da nije član niti jedne grupe te mu se daje ikona koja ga potiče da se pridruži grupi. Prozor koji korisniku omogućuje unos naziva grupe i uspostavljanje nove grupe pojavljuje se kada se klikne na plus ikonu. Klasa „DatabaseService“ obavlja potrebne zadatke kao odgovor na uspostavu grupe, kao

[20] Logička vrijednost (boolean) koja označava istinu.

što je pohranjivanje podataka grupe i ažuriranje popisa korisničke grupe. Početna stranica također ima ladicu, kojoj se može pristupiti dodirivanjem ikone na traci aplikacije, kako bi se ponudile dodatne mogućnosti navigacije. Ladica na vrhu prikazuje korisničko ime i profilnu sliku. Ladica sadrži poveznicu za stranicu profila gdje korisnici mogu vidjeti svoje osobne podatke i poveznicu za odjavu za izlaz iz programa. Korisnici su spriječeni od nenamjerne odjave zahvaljujući okviru za potvrdu koji se pojavljuje kada odaberu "Log out".

4.6.4. Datoteka „profile_page.dart“

Stranica profila korisnicima pruža personalizirani prostor za pregled informacijama o profilu. Nudi korisničko sučelje s raznim odjeljcima koji prikazuju bitne pojedinosti poput korisničkog imena i e-pošte. Ime i e-mail korisnika prikazani su u dijelu koji se nalazi ispod slike profila. Ova konfiguracija osigurava vidljivost identifikacije i kontaktnih podataka korisnika. Stranica profila koristi separator za vizualno razlikovanje različitih komponenti. Korisnicima će biti lakše razumjeti prikazane informacije zahvaljujući ovom separatorom, vodoravnoj liniji koja vizualno odvaja ime korisnika od adrese e-pošte.

4.6.5. Datoteka „search_page.dart“

Stranica za pretraživanje omogućuje korisnicima pretraživanje grupa unutar aplikacije. Nudi jednostavno sučelje gdje korisnici mogu unijeti svoj upit za pretraživanje i vidjeti rezultate pretraživanja u stvarnom vremenu. Glavni sadržaj stranice za pretraživanje sastoji se od polja za unos teksta i gumba za pretraživanje. Polje za unos teksta omogućuje korisnicima unos upita za pretraživanje grupa. Gumb za pretraživanje, predstavljen ikonom povećala, pokreće funkciju pretraživanja kada se klikne. Postoji dio koji prikazuje rezultate pretraživanja ispod obrasca za unos pretraživanja. Korisnički rezultati pretraživanja prikazuju se u obliku popisa. Pločica grupe koja predstavlja svaki rezultat pretraživanja prikazuje ključne pojedinosti o

grupi, uključujući njezin naziv, administratora i popratni avatar. Pločica grupe ima kružni avatar i oblikovan tekst kako bi bila vizualno privlačna. Svaka pločica grupe nudi korisniku priliku da se pridruži ili napusti grupu. Gumb za pridruživanje ili izlazak iz grupe, pokreće povezanu funkciju kada se klikne. Ovisno o korisnikovom statusu članstva u grupi, izgled gumba se dinamički mijenja, s oznakom "Joined" koja se pojavljuje ako je korisnik već član. Kako bi se poboljšalo korisničko iskustvo, stranica za pretraživanje uključuje indikator učitavanja, predstavljen kružnim indikatorom napretka, koji se prikazuje kada se dohvaćaju rezultati pretraživanja. Ovaj indikator obavještava korisnika da aplikacija aktivno obrađuje njihov zahtjev za pretraživanje.

4.6.6. Datoteka „chat_page.dart“

Stranica za *chat* korisnicima pruža platformu za sudjelovanje u razgovorima u stvarnom vremenu unutar određene grupe. Nudi korisničko sučelje na kojem korisnici mogu pregledavati i slati poruke drugim članovima grupe. Naziv trenutnog grupnog razgovora pojavljuje se kao naslov na traci aplikacije. Naslov se automatski mijenja prema odabranoj grupi. Osim toga, korisnici mogu, klikom na ikonu na desnoj strani alatne trake, pristupiti informacijama o grupi gdje mogu pročitati detalje o grupi, uključujući njezinog administratora i članove. Okvir za prikaz poruka i područje za unos poruke čine najveći dio glavnog sadržaja stranice. „ListView“ se koristi za prikaz povijesti *chat-a*. Oblačić s porukom, koja uključuje pojedinosti o pošiljatelju i sadržaju poruke, koristi se za prikaz svake poruke u povijesti razgovora. Pomoću podataka o razgovorima koji su sada dostupni, oblačići poruka se dinamički prikazuju. Jasna komunikacija tijekom rasprava omogućena je vizualnom razlikom između poruka koje šalje trenutni korisnik i onih koje šalju drugi korisnici. Pri dnu stranice nalazi se okvir za unos poruke. Na njemu se nalazi okvir za unos teksta i gumb za slanje. Korisnici mogu upisivati svoje poruke u polje za unos teksta, a klikom na gumb za slanje aktivira se funkcija slanja. Kako bi se olakšala komunikacija u stvarnom vremenu, stranica za razgovor koristi mehanizam strujanja. Poruke se dohvaćaju i prikazuju pomoću „StreamBuildera[21]“, koji prati ažuriranja iz *stream-a*[22] razgovora. Kako se

[21] Widget u Flutteru koji se koristi za izgradnju korisničkog sučelja na temelju podataka koje isporučuje *Stream*.
[22] Koncept u programiranju koji se koristi za asinkrono slanje i primanje podataka ili događaja.

nove poruke šalju ili primaju, tok *chat-a* pruža trenutna ažuriranja, osiguravajući da korisnici imaju pristup najnovijim porukama u stvarnom vremenu. Ime administratora grupe također je prikazano na stranici za razgovor. Ime administratora stranice za razgovor izvlači se iz baze podataka. Korisnici mogu koristiti ove informacije kako bi locirali i stupili u kontakt s administratorom grupe.

4.6.7. Datoteka „group_info_page.dart“

Stranica s informacijama o grupi pruža korisnicima bitne informacije o određenoj grupi. Omogućuje korisnicima pregled pojedinosti o grupi, kao što je njezin naziv, ime administratora i popis članova. Dodatno, nudi mogućnost napuštanja grupe. Nakon pristupa stranici, korisnicima se prikazuje traka na vrhu. Traka sadrži ime grupe i ikonu za izlaz iz grupe. Klikom na ovu ikonu pokreće se dijaloški okvir za potvrdu, osiguravajući da korisnici imaju priliku potvrditi svoju odluku prije napuštanja grupe. Glavni sadržaj stranice podijeljen je u dva dijela. Osnovne informacije o grupi predstavljene su u prvom dijelu na vizualno ugodan način. Ima kružni avatar koji predstavlja grupu zajedno s tekstom koji navodi administratora i naziv grupe. Ime administratora preuzeto je iz priložene opcije „adminName“, dok se naziv grupe dinamički dobiva iz baze podataka i prikazuje na vidljivom mjestu. Popis članova grupe nalazi se u drugom odjeljku stranice. Za izradu ovog popisa koristi se „StreamBuilder“. Kružni avatar i korisničko ime člana prikazani su na „ListTile-u“ koji služi kao prikaz svakog člana na popisu. Korisničko ime dobiva se iz podataka o članovima koji su dohvaćeni i prikazani pored odgovarajućeg avatara. Ako nema članova, prikazuje se obavijest koja pokazuje da je grupa prazna. Kako bi se poboljšalo korisničko iskustvo, stranica koristi indikator učitavanja dok se podaci o članovima dohvaćaju. Ovaj indikator osigurava da su korisnici svjesni da stranica dohvaća informacije i sprječava bilo kakvu zabunu ili frustraciju uzrokovanu praznim zaslonom tijekom procesa učitavanja.

5. Zaključak

Aplikacija za čavrljanje koju sam prikazao u ovom radu primjer je snage modernog razvoja mobilnih aplikacija. Ovaj rad prikazuje ogroman potencijal modernih komunikacijskih tehnologija. Iskorištavanjem snage mobilnih platformi i iskorištavanjem usluga temeljenih na oblaku, aplikacija korisnicima nudi pojednostavljen i prikladan način povezivanja i interakcije s drugima. Putem značajki kao što su sigurna prijava i registracija, intuitivna korisnička sučelja, razmjena poruka u stvarnom vremenu i grupna funkcionalnost, aplikacija olakšava besprijekornu komunikaciju i suradnju. Nadalje, integracija *Firebase-a* pruža pouzdanu i skalabilnu pozadinsku infrastrukturu. Kako tehnologija napreduje, aplikacije za čavrljanje kao što je ova nastavit će redefinirati način na koji komuniciramo i stvaramo odnose u digitalnom dobu, nadilazeći geografske granice i omogućavajući smislene veze na globalnoj razini. U konačnici, ova aplikacija primjer je dubokog utjecaja tehnologije na modernu komunikaciju i naglašava sve veću važnost poticanja digitalne povezanosti i društvenog angažmana.

Reference

n.d. *Firebase documentation.* Pokušaj pristupa 5. Lipanj 2023. https://firebase.google.com/docs.

n.d. *Flutter documentation.* Pokušaj pristupa 5. Lipanj 2023. https://docs.flutter.dev/.

freecodecamp. 2022. *YouTube.* 24. Veljača. Pokušaj pristupa 2. Svibanj 2023. https://www.youtube.com/watch?v=VPvVD8t02U8.

—. 2023. *YouTube.* 12. Travanj. Pokušaj pristupa 4. Lipanj 2023. https://www.youtube.com/watch?v=UFD4SP91tSM.

n.d. *GeeksforGeeks.* Pokušaj pristupa 10. Svibanj 2023. https://www.geeksforgeeks.org/flutter-tutorial/.

n.d. *JavatPoint.* Pokušaj pristupa 11. Svibanj 2023. https://www.javatpoint.com/flutter.

n.d. *Tutorials Point.* Pokušaj pristupa 6. Svibanj 2023. https://www.tutorialspoint.com/firebase/.

n.d. *Wikipedia.* Pokušaj pristupa 4. Lipanj 2023. https://en.wikipedia.org/wiki/Flutter_(software).

n.d. *Wikipedia.* Pokušaj pristupa 4. Lipanj 2023. https://en.wikipedia.org/wiki/Firebase.

Popis slika

Popis priloga

Programski kod aplikacije	https://github.com/DBDoco/Chat-App
Izgrađena verzija aplikacije	https://mega.nz/file/0cliRD4A#M1EXUr8MMF1UzgBrlBOihjZok99LWYUzCNdD8s5TvC0
Demo video rada aplikacije	https://www.youtube.com/watch?v=cWVPWImupUE