Mobilna chat aplikacija

Jurišić, Ivan

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

Sveučilišni studij

MOBILNA CHAT APLIKACIJA

Diplomski rad

Ivan Jurišić

Osijek, 2022.

SADRŽAJ

1. UVOD

Sa sve većim razvojem tehnologije dolazi do porasta komunikacije između ljudi. Sve je veća potražnja za mobilne aplikacije pa se tako u Google play trgovinu svakim danom doda preko 3500 novih aplikacija. Mnoge od tih aplikacija se koriste za razne usluge gdje je komunikacija sa korisnicima nužna. Cilj ovog rada je stvaranje aplikacije za dopisivanje. Primarna funkcija je omogućiti korisnicima da šalju tekstualne poruke jedni drugima uz mogućnost slanja multimedijskih sadržaja poput slike ili datoteke. Iako će ovaj rad poslužiti kao uvid kako napraviti aplikaciju za dopisivanje ovakve aplikacije mogu poslužiti kao moduli nekim drugim aplikacijama koje korisnici često koriste. Na primjer, aplikacije koje služe kao e-trgovine moraju imati neku vrstu korisničke podrške, a jedan od najboljih načina je omogućiti korisnicima da kontaktiraju korisničku podršku putem poruka.

Prvo poglavlje obuhvaća teoriju koja objašnjava izvorne i višeplatformske alate i programska okruženja koja su bitna za shvatiti kako bi se razumjela potreba za višeplatformskim aplikacijama navedena su i kratko objašnjena postojeća rješenja mobilnih chat aplikacija. Drugo poglavlje opisuje korištene tehnologije i kratki opis svaku od korištenih tehnologija. Treće poglavlje opisuje način razvoja aplikacije. U četvrtom i zadnjem poglavlju prikazan je izgled Flutter aplikacije svaki zaslon je objašnjen pomoću slike.

1.1. Zadatak diplomskog rada

Potrebno je napraviti mobilnu aplikaciju koja će omogućiti korisnicima razmjenu tekstualnih poruka, prijenos slika i datoteka. Kao glavna tehnologija koristit će se Flutter programsko okruženje.

2. RAZVOJNO OKRUŽENJE

Kroz ovo poglavlje će se istražiti najpopularnije tehnologije i načini za izradu mobilnih aplikacija. Glavni cilj ovog rada je izrada višeplatformske aplikacije, stoga ima smisla istražiti koje opcije postoje, koje su najpopularnije i koje su najbolje.

Istraživanje je podijeljeno na tri kategorije:

- Izvorne (engl. *native)* tehnologije za razvoj aplikacija.
- Web tehnologije za razvoj aplikacija.
- Višeplatformske tehnologije za razvoj aplikacija.

Za izvorno programiranje android mobilnih uređaja najčešće se koristi Android Studio IDE u kojem se koriste programski jezici Java i Kotlin**.** Kod iOS uređaja koristi se Xcode IDE u kojem se najčešće koristi Swift programski jezik.

Web programiranje je staro koliko je i internet. Posljednjih nekoliko godina razvijaju se sve više popularniji web programski okviri neki od njih su : Angular, React, Django, Vue.js, Laravel**,** a većina ih koristi programske jezike poput: JavaScript, Typescript, PHP-a u kombinaciji sa HTML5 i CSS3.

Kod višeplatformskih tehnologija u posljednjih nekoliko godina istaknula su se dva giganta. Flutter SDK kojeg podupire Google i React Native razvojni okvir kojeg podupire Meta Platforms (nekadašnji Facebook).

Postoji još puno tehnologija koje nisu navedene i teško ih je kategorizirati a za potrebe ovog rada u daljnjem tekstu spominjat će se samo najbitnije tehnologije za razvoj mobilnih aplikacija.

2.1.Izvorni razvoj mobilnih aplikacija

Postoji puno prednosti izvornog razvoja mobilnih aplikacija. Briljantno korisničko iskustvo je zagarantirano kod izvorno napisanih aplikacija pošto su performanse takvih aplikacija najbolje u klasi. Izvorno programiranje omogućuje korisniku da stvara sučelja koja su primjerena platformi za koju se izrađuju. Na taj način korisnici prilikom korištenja aplikacija imaju poznato iskustvo koje očekuju i prepoznaju u ostalim aplikacijama koje koriste jer su dizajnirane pomoću iste platforme. Također, izvorne tehnologije su obično popraćene već nekoliko desetljeća starim i popularnim programskim jezicima poput Jave i Objective C-a, a u skorije vrijeme i njihovim

nasljednicima Kotlin i Swift. iz tog razloga vrlo je lako pronaći gotove primjere kôda ustanovljene najbolje prakse iz raznih izvora te ogromnu zajednicu programera koja stoji iza njih. Najveći nedostatak izvornih aplikacija je to što se gotovi proizvodi mogu plasirati samo za platforme koje su im izvorne odnosno android aplikacije pokrećemo na android uređajima dok iOS aplikacije na iOS uređajima. Dakle potrebno je napraviti istu aplikaciju za više različitih platformi što ovisno o zahtjevima aplikacije može biti izrazito skupo i dugotrajno.

U daljnjem tekstu biti će opisane dvije najpopularnije tehnologije za razvoj izvornih mobilnih aplikacija: Android studio korišten za razvoj aplikacija na Android platformi i Xcode koji je korišten za razvoj aplikacija na iOS platformi.

2.1.1. Android studio

Android studio je službeno razvojno okruženje (IDE) za razvoj Googleov-ih Android operativnih sustava temeljen na IntelliJ IDEA softveru. Dostupan je za Windows, macOS i Linux operativne sustave. Namijenjen je kao zamjena Eclipse Android Development Tools-u kao primarno razvojno okruženje za razvoj izvornih android aplikacija [2].

Glavne značajke Android studio alata:

- Fleksibilan sustav izgradnje aplikacija zasnovan na Gradle-u.
- Brz i značajkama bogat emulator.
- Jedinstveno okruženje za razvoj aplikacija namijenjenih android uređajima.
- Primjena promjena tokom izgradnje aplikacije bez resetiranja aplikacije.
- Predlošci kôda i GitHub integracija koja pomaže izgradnji značajki.
- Opsežni alati i okviri za testiranje.
- Lint alati za promatranje performansi, upotrebljivosti, kompatibilnosti verzija i drugih problema.
- Podrška za C++ i NDK podrška.
- Ugrađena podrška za Google-ov oblak.

Android studio je prvi puta predstavljen 16. svibnja 2013, dok je prva stabilna verzija izašla u prosincu 2014 godine. 07. svibnja 2019. Kotlin je zamijenio Javu kao Google-ov preferirani jezik za razvoj android aplikacija. Java i C++ su i dalje podržani. U trenutku pisanja ovog rada najnovija inačica je Android studio Dolphin [3] .

2.1.2. Xcode

Xcode je Apple-ovo integrirano razvojno okruženje (IDE) za macOs operativne sisteme koje se koristi za razvoj softvera za macOS, iOS, iPadOS, watchOS i tvOS. Prvi puta je objavljen krajem 2003. godine. Xcode podržava programske jezike: C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, Rez i Swift. Apple snažno promovira Swift kao svoj jezik izbora na svim svojim platformama, a ključni element ovoga je da su SwiftUI aplikacije izvorne na svim platformama u vlasništvu Apple-a. Korisnici mogu jednostavno presaditi osnovnu logiku SwiftUI iPhone aplikacije u macOS aplikaciju i obrnuto. Swift, SwiftUI i Xcode rade zajedno kao jedan. Swift je prvi puta objavljen 2014 i namijenjen je kao zamjena za Objective C. Swift je moderan i relativno mlad jezik sa komparativno čistom i lako čitljivom sintaksom. Sadrži odlične mogućnosti upravljanja memorijom što smanjuje količinu potrebne memorije u aplikacijama. SwiftUI omogućuje programerima da izrade korisnička sučelja za sve Apple-ove platforme [4].

2.2. Višeplatformski razvoj mobilnih aplikacija

Višeplatformski razvoj za cilj ima stvaranje jedne aplikacije koja radi identično na više platformi. Koristi tehnologije neovisne o platformi kao što su HTML, CSS i JavaScript. Višeplatformski način razvoja aplikacija ima mnogo prednosti kao što su :

- Izrada zasebnih izvornih aplikacija za svaku platforma je skupa, dok višeplatformska aplikacija koristi jedan zajednički kôd, što pomaže tvrtkama da ostanu u okvirima svojih proračuna.
- Troškovi su smanjeni jer je za razvoj i održavanje aplikacije potreban samo jedan tip programera.
- U većini slučajeva potrebno je znanje standardnih jezika, dok veliki dio posla obavljanju razvojni alati.
- Višeplatformske aplikacije imaju izvorni izgled i dojam što korisnicima nudi odlično iskustvo.
- Lakše je privući velik broj korisnika različitih mobilnih uređaja uz nižu cijenu.

Međutim, postoje neki problemi s kojima se suočavaju :

- Složenije aplikacije ponekad kombiniraju izvorne i strane komponente što može utjecati na performanse.
- Višeplatformske aplikacije ne mogu podržavati izvorne funkcije i značajke mobilnih uređaja kao što su napredna grafika i animacija ili 3D efekti.
- Ponekad to rezultira ograničenom funkcionalnošću i lošijim dizajnom aplikacije.
- Kada Google i Apple dodaju nove značajke na Android i iOS platforme izvorna rješenja mogu ih odmah početi koristiti ali višeplatformske aplikacije moraju pričekati dok se ova ažuriranja ne prilagode odabranom okviru za više platformi. Stoga će uvijek doći do kašnjenja ažuriranja.

Iako imaju svoje prednosti i ograničenja višeplatformski način razvoja aplikacija je sve popularniji a neki od najpopularnijih razvojnih okruženja su Flutter, React Native, Cordova, Ionic, Xamarin, NativeScript, PhoneGap i mnogi drugi.

2.3. Flutter

Flutter je Googleov prijenosni UI alat za izradu prekrasnih, izvorno kompajliranih (eng. *Compiled*) aplikacija za više platformi iz jedne baze koda. Flutter se razlikuje od ostalih razvojnih okvira za razvoj mobilnih aplikacija po tome što se on ne omotava oko sistemskih funkcija platforme kao neki programski okviri koji su napisani pomoću JavaScript programskog jezika. Flutter koristi svoj stroj za interpretaciju programskog kôda koji direktno koristi resurse platforme umjesto SDK platforme na kojoj se izvršava Flutter aplikacija. Flutter je izgrađen pomoću C, C++, Dart i Skia (*engine* za 2D iscrtavanje). Kod android uređaja C i C++ kompajlirani s Androidovim SDK-om. Dart kôd (i od SDK i napisan) se prije vremena (AOT) kompiliraju u izvorne, ARM i x86 biblioteke. Te su biblioteke uključene u "runner" Android projekta, a na taj način dobijemo .apk datoteku. Kada se pokrene aplikacija učitava biblioteku Flutter. Svako iscrtavanje komponenti, unos ili primjerice rukovanje događajima delegira se kompajliranom Flutteru i kôdu aplikacije što je slično načinu na koji radi puno igara. Kod iOS uređaja C i C++ kôd su kompajlirani pomoću LLVM-a. Dart kôd (i od SDK i napisan) se prije vremena (AOT) kompiliraju u izvornu ARM biblioteku. Ta je biblioteka uključena u "runner" iOS projekt, a na taj način dobijemo .ipa datoteku [5].

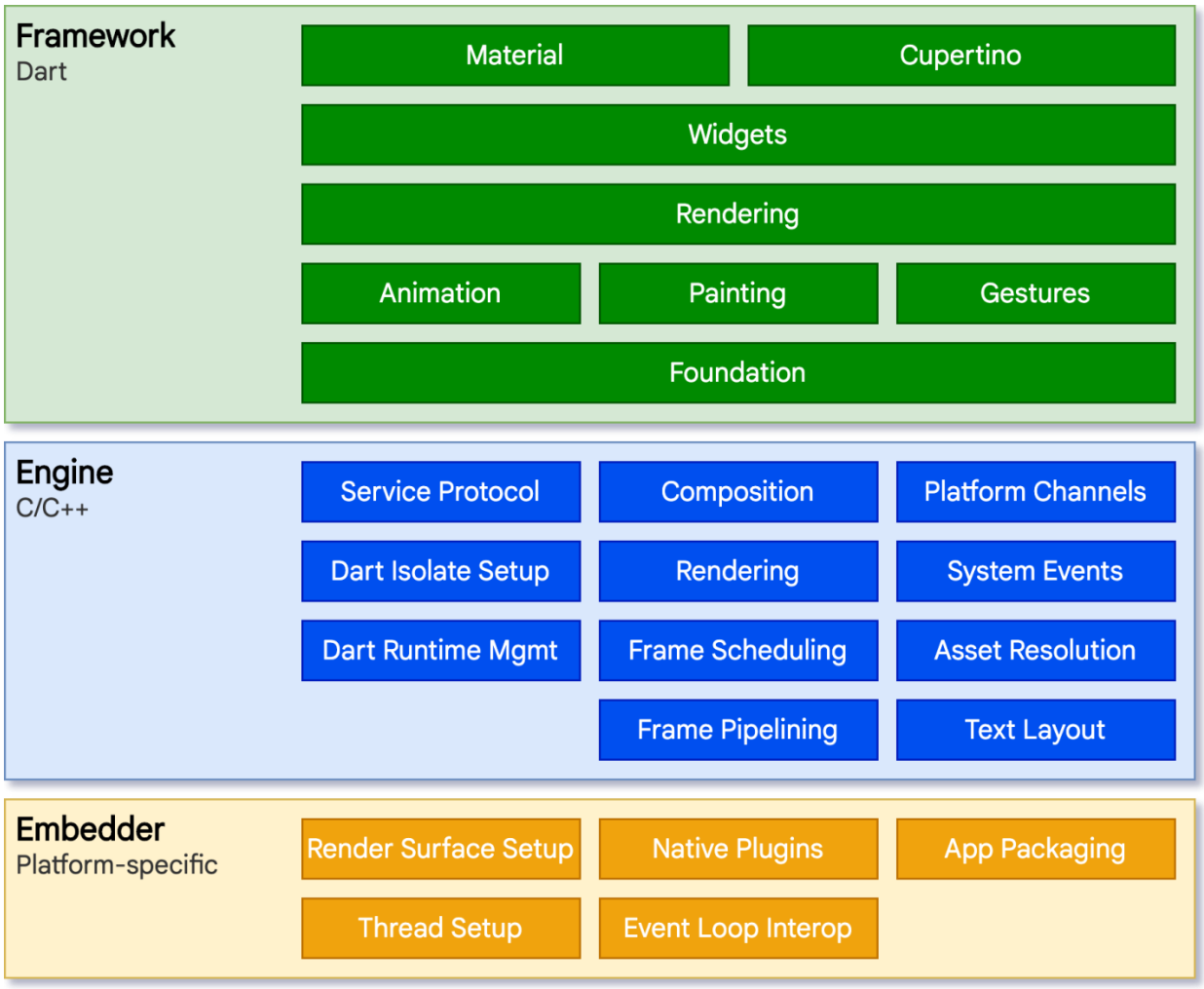
Hot reload i hot restart

Flutterova *hot reload* značajka omogućava brzo i jednostavno eksperimentiranje, izradu korisničkog sučelja, dodavanje novih značajki i lagano ispravljanje grešaka. Funkcionira na način da ubacuje ažurirane datoteke izvornog kôda u pokrenutu Dart virtualnu mašinu (eng. *Dart virtual Machine VM)*[6]. Nakon što VM ažurira klase s novim verzijama polja i funkcija Flutter razvojni okvir automatski ponovno gradi stablo widgeta omogućujući brz pregled promjena u aplikaciji bez izmjene stanja widgeta.

Hot restart značajka učitava promjene kôda u VM i ponovno pokreće Flutter aplikaciju ali gubi stanje aplikacije.

2.3.1. Slojevitost arhitekture

Flutter je dizajniran kao slojevit i proširiv sustav. Sačinjava ga niz neovisnih biblioteka od kojih svaka ovisi o temeljnom sloju. Slojevi iznad nemaju privilegirani pristup slojevima ispod, a svaki dio razine okvira dizajniran je tako da bude zamjenjiv. Slojevitost arhitekture prikazana je na slici 2.1.



Sl. 2.1. Arhitektura Flutter programskog okvira

Za temeljni operacijski sustav Flutter aplikacije su napravljene na isti način kao i svaka druga izvorna aplikacija. Embedder pruža ulaznu točku i koordinira sa operativnim sustavom kako bi pristupao uslugama za iscrtavanje, pristupačnost i unos te upravlja petljom događaja za poruke. Embedder je napisan u jeziku koji je prilagodljiv svakoj platformi. Za Android se koriste Java i C++, za iOS i macOS se koriste Objektivni C i Objektivni C++. Flutter koristi embedder kako bi integrirao Dart kôd u aplikaciju kao modul, ali on ujedno može biti i cijeli sadržaj aplikacije. Flutter uključuje brojne embeddere za uobičajene ciljne platforme, ali postoje i drugi embedderi.

U jezgri se nalazi Flutter stroj koji je napisan na C++ i podržava primitivne tipove potrebne za podršku svim Flutter aplikacijama. Stroj je odgovoran za rasteriziranje složenih scena kad god je

potrebno slikati novi okvir. Pruža implementaciju Flutter jezgrenog API-ja na niskoj razini uključujući grafiku (kroz Skia), izgled teksta, datoteke i mrežni I/O, podršku za pristupačnost, arhitekturu dodataka i Dart vrijeme izvođenja i kompajliranje alata [7].

2.3.2. Dart

Dart je programski jezik dizajniran za razvoj web i mobilnih aplikacija. Može se koristiti i za izradu poslužiteljskih i desktop aplikacija. To je objektno orijentiran jezik koji sadrži klase i skuplja smeće sa sintaksom u stilu C-a. Može se kompajlirati u izvorni kôd ili JavaScript i podržava sučelja, apstraktne klase, reificirane generike i zaključivanje tipova [8].

Flutterov tim je odlučio učiniti Dart kao primarni SDK za njihov razvojni okvir i widgete. Utvrdili su da mnogi jezici ispunjavaju neke zahtjeve od potreba autora razvojnog okvira do programera i u konačnici krajnjih korisnika no Dart je postigao visoke ocjene na svim dimenzijama evaluacije i ispunio zahtjeve i kriterije.

Vrijeme izvođenja Darta kôda i kompajler omogućuju kombinaciju dvije kritične značajke za Flutter :

- Brzi razvojni ciklus baziran na JIT-u koji omogućuje promjenu oblika i ponovno učitavanje stanja na jeziku s tipovima.
- Plus kompajler prije vremena koji emitira učinkovit ARM kôd za brzo pokretanje i predvidljiva izvedba proizvodnje.

Neke od značajki Darta su :

- Produktivnost programera – štedi inženjerske resurse dopuštajući programerima da kreiraju aplikacije za iOS i Android istom bazom kôda.
- Objektno orijentiran je – velika većina programera ima iskustva s objektno orijentiranim razvojem što olakšava učenje Darta i Flutter razvojnog okruženja.
- Predvidljivost, visoke performanse – s Flutterom je omogućeno stvaranje brzih, fluidnih korisničkih sučelja.
- Brza alokacija – Flutter razvojno okruženje koristi tijek funkcionalnosti stila koji uvelike ovisi o temeljnom alokatoru memorije koji učinkovito rukuje malim, kratkotrajnim alokacijama.

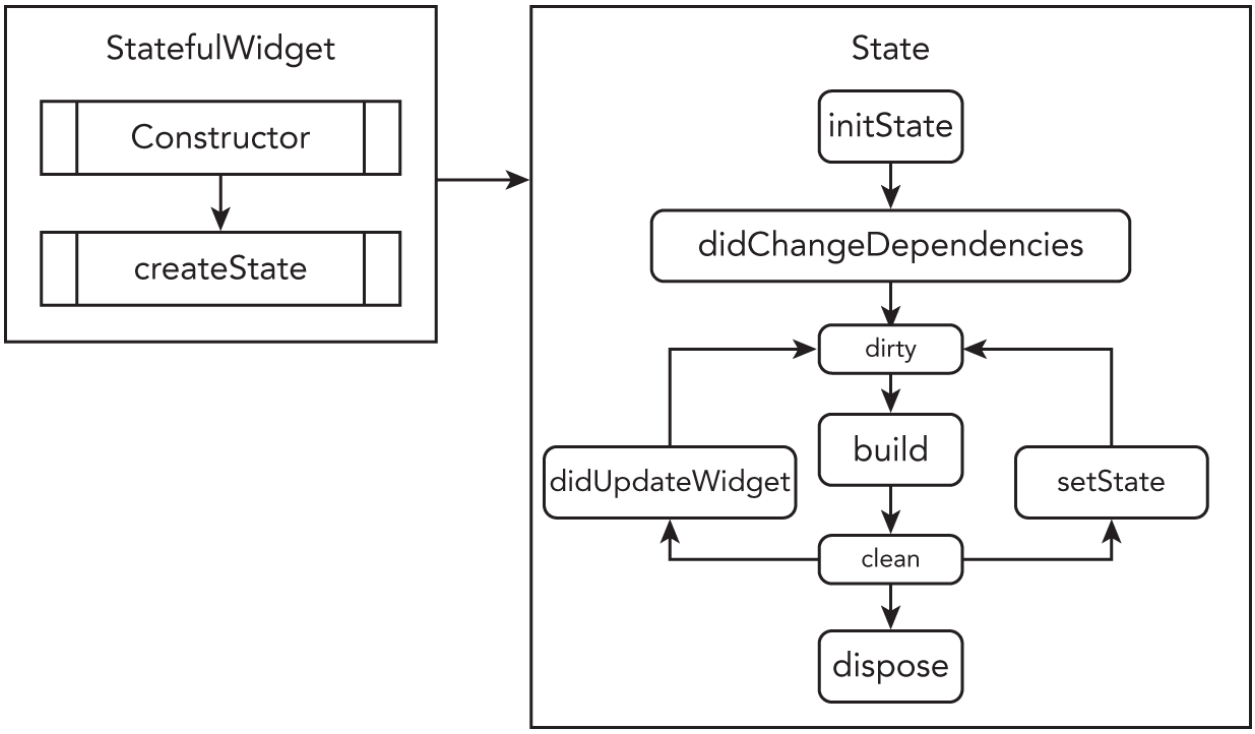
Za konkretnu izradu aplikacija Flutter se oslanja na korištenje velikog asortimana već gotovih widgeta. Glavna ideja je da se korisničko sučelje gradi od widgeta u obliku stabla.

2.3.3. Životni ciklus Widgeta

Korisničko sučelje je ažurirano kao odgovor na događaje te na taj način poručuje razvojnom okviru da zamjeni widget sa drugim widgetom. Flutter potom uspoređuje nove widgete sa starim i ažurira korisničko sučelje.

Neki widgeti nemaju promjenjivo stanje (na primjer ikona ili oznaka).

Kada se jedinstvene karakteristike widgeta moraju mijenjati na temelju određenih čimbenika, tada widget ima stanje. Na primjer, kada korisnik dodirne gumb koji služi kao brojač tada vrijednost brojača označava stanje za taj widget. Kada se vrijednost na widgetu promjeni widget se treba ponovno izgraditi i ažurirati svoj dio korisničkog sučelja. Ovi widgeti podklase *StatefulWidget* (jer je sam widget nepromjenjiv) pohranjuju promjenjivo stanje u zasebnu klasu koja je pod klasa State. StatefulWidget-i nemaju *build* metodu.



Sl. 2.2. Životni ciklus *StatefulWidget* klase

Glavna razlika između Stateful i Stateless widgeta je ta da se kod Stateful widgeta ne okida događaj *build* nego se okida događaj *createState.* Kada se okine događaj *createState* za taj Widget se stvara objekt stanja. Kod takvih widgeta se događaj *build* odvija u njegovom objektu stanja.

Životni ciklus temelji se na stanju i načinu na koji se ono mijenja. Widget s praćenjem stanja ima stanje tako da na temelju njega možemo objasniti životni ciklus widgeta. Faza životnog ciklusa:

- *createState().*
- *initState().*
- *didChangeDependencies().*
- *build().*
- *didUpdateWidget().*
- *setState().*
- *dispose().*

initState() – Metoda koja se koristi pri stvaranju klase sa stanjem. Mogu se inicijalizirati varijable, svojstva i sl.

createState() – Kada se kreira widget sa stanjem Flutter poziva metodu *createState()* i mora se nadjačati.

build() – Metoda koja se koristi svaki puta kada se Widget ponovno izgradi. To se može dogoditi kod pozivanja *initState, didChangeDependencies, didUpdateWidget* ili kada se stanje promjeni putem poziva *setState*.

didChangeDependencies() – Metoda koja se poziva odmah nakon *initState* i kada se ovisnost objekta *State* promjeni putem naslijeđenog Widgeta.

didUpdateWidget() – Metoda koja se poziva svaki puta kada se promjeni konfiguracija widgeta. Tipičan slučaj je kada roditelj prosljeđuje neku varijablu widgetu djetetu preko konstruktora.

dispose() – Metoda koja oslobađa resurse aplikacije na način da obriše objekt sa stanjem.

Životni ciklus komponenti koje se koriste tokom izrade aplikacije samo je jedna od razlika platformi Android i iOS. Životni ciklusi kod mobilnih platformi odnose se na komponente cijelog grafičkog sučelja jednog zaslona, a kod Fluttera se svaka komponenta koja se nalazi unutar hijerarhije grafičkog sustava može ali i ne mora imati životni ciklus. Životni ciklus se ne dijeli između komponenti. Na taj način postiže se ponovno iscrtavanje elemenata koji su promijenili svoje stanje.

2.4. Pregled postojećih rješenja

WhatsApp

Većina ljudi je vjerojatno već čula ili koristi WhatsApp, i to s dobrim razlogom. WhatsApp omogućuje besplatne glasovne pozive, video pozive i slanje izravnih poruka, a sve putem intuitivnog sučelja na iOS i Android uređajima. Osim brojnih značajki koje WhatsApp omogućuje tu je i mogućnost uparivanja mobilnog uređaja sa web platformom [10].

Viber

Viber je višeplatformska softverska aplikacija za slanje poruka u vlasništvu Japanske kompanije Rakuten. Korisnici se u ovoj aplikaciji identificiraju putem broja mobilnog telefona, iako je usluga dostupna i na računalima. Od 2018. postoji više od milijardu registriranih korisnika na mreži [11].

BlaBlaCar

Aplikacije za dopisivanje, glasovni ili video razgovor ne moraju nužno imati jednu svrhu. Primjer aplikacije u kojoj je dopisivanje korisnika sekundarna funkcionalnost je BlaBlaCar. BlaBlaCar je web stranica i mobilna aplikacija koja povezuje vozače i putnike koji su spremni putovati zajedno između gradova i podijeliti troškove putovanja.

3. KORIŠTENE TEHNOLOGIJE

U ovome poglavlju objašnjene su tehnologije, biblioteke, usluge i alati koji su korišteni za izradu aplikacije.

3.1. Programski alati i tehnologije

U prvom poglavlju spomenuto je kako će se aplikacija raditi u programskom sučelju Flutter i Android Studio alatu.

3.1.1. Firebase

Firebase je alat za kreiranje web i mobilnih aplikacija razvijena od strane Google-a. Firebase je evoluirao iz Envolvea prethodnog start-upa kojeg su osnovali James Tamplin i Andrew Lee 2011. Envolve je programerima omogućio API koji omogućuje integraciju funkcionalnosti online chata u njihove web stranice. Nakon što su pustili chat uslugu, Tamplin i Lee su otkrili da se ona koristi za prosljeđivanje podataka aplikacije koji nisu chat poruke. Programeri su koristili Envolve za sinkronizaciju podataka aplikacije kao što je stanje igre u stvarnom vremenu među svojim korisnicima. Tamplin i Lee odlučili su odvojiti chat sustav i arhitekturu u stvarnom vremenu koja ga je pokretala. Osnovali su Firebase kao zasebnu tvrtku 2011. godine, a javnosti je predstavljena u travnju 2012. godine [13].

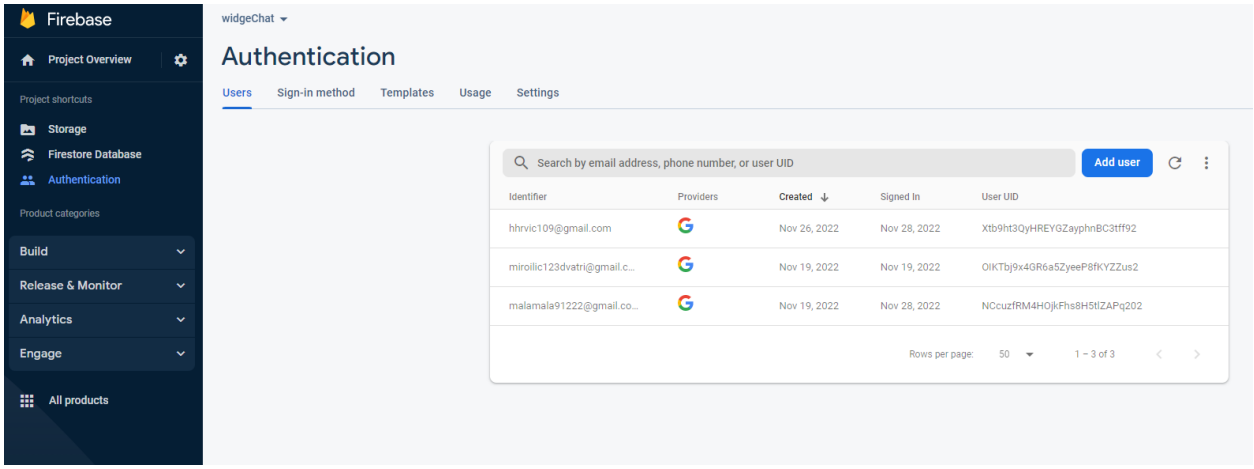
Neke od usluga pruža su:

- Baza podataka u stvarnom vremenu (eng. *Real time database)* – omogućuje spremanje i sinkronizaciju podataka sa aplikacijom u stvarnom vremenu.
- Provjera identiteta (eng. *Authentification*) – omogućuje provjeru identiteta korisnika aplikacije.
- Firestore baza podataka (eng. *Database)* – Podaci se spremaju u oblak (eng. *Cloud*) u JSON formatu.

3.1.2. Firebase provjera identiteta

Za većinu aplikacija potrebno je znati identitet korisnika. Firebase provjera identiteta pruža tu mogućnost. Poznavanje identiteta korisnika omogućuje korisniku da sigurno sprema svoje podatke

u oblak i pruža personalizirano iskustvo. Podržava provjeru autentičnosti pomoću lozinki, telefonskih brojeva, popularnih pružatelja identiteta poput *Facebook* socijalne platforme, *Google* korisničkog računa, *Twitter* socijalne platforme i sl. Firebase provjeru identiteta moguće je koristiti sa više različitih tehnologija i različitih platformi.



Sl. 3.1. Životni ciklus *StatefulWidget* klase

3.1.3. Firebase baza podataka

Firebase baza podataka se nalazi u oblaku. Podaci se pohranjuju kao JSON i sinkroniziraju u stvarnom vremenu sa svakim povezanim klijentom. Ova značajka Firebase-a također je neovisna o tehnologiji uz koju se. Svi klijenti dijele jednu instancu baze podataka u stvarnom vremenu i automatski primaju ažuriranja s najnovijim podacima.

Ono što Firebase bazu podataka čini dobrom je fleksibilnost, odnosno, njezin model podataka. Podaci se spremaju u dokumente, a dokumenti u kolekcije. Kolekcije služe kao kontejneri za dokumente, a dokumenti služe za zapisivanje i organiziranje podataka. Dokumenti mogu sadržavati svoje kolekcije odnosno pod kolekcije.

Primjer strukture u firebase bazi podataka :

- Korisnici (kolekcija)
  - korisnik_1 (dokument)
    - ime (podatak)
    - prezime (podatak)
    - e-mail (podatak)
    - ...
    - lista_prijatelja (pod kolekcija)
      - prijatelj_1 (dokument)
        - podatak_1 (podatak)
        - ...

Podatke iz Firebase baze podataka dohvaćamo upitima. Može se dohvatiti dokument ili cijela kolekcija. Upitima je moguće manipulirati podatke na razne načine poput filtriranja, sortiranja, limitiranja i sl.

3.1.4. Najbitniji korišteni paketi

Image picker –paket za Flutter razvojno okruženje za odabir slika iz galerije i snimanje novih slika fotoaparatom na Android i iOS mobilnim uređajima.

Google sign in – paket za Google prijavu. Siguran sustav provjere identiteta za prijavu s Google računom na Android i iOS mobilnim uređajima.

FlutterToast – paket za jednostavno kreiranje *toast* poruka u jednom retku kôda.

SharedPreferences – Paket za čitanje i pisanje jednostavnih ključ-vrijednost (eng. *key-value*) parova. Podaci se mogu zadržati na disku asinkrono i nema jamstva da će se zapisi zadržati na disku nakon povratka, tako da se ovaj podatak ne bi trebao koristiti za pohranu kritičnih podataka.

Path provider – Paket za pronalaženje često korištenih lokacija u datotečnom sustavu. Podržava Android, iOS, Linux, macOS i Windows.

File picker – Paket koji omogućuje korištenje izvornog pretraživača datoteka za odabir jedne ili više datoteka uz podršku za filtriranje proširenja.

4. IZRADA MOBILNE APLIKACIJE

Uz pomoć službene dokumentacije uz dijelove izrađene po uzoru na postojeće primjere sličnih aplikacija poput *WhatsApp, Messenger* i sličnih, aplikacija se tijekom procesa izrade kontinuirano razvijala i nadograđivala novim programerskim praksama, tehnologijama i programerskim alatima.

4.1. Izgradnja zaslona za prijavu

```
class ChatUser extends Equatable {
  final String id;
  final String photoUrl;
  final String displayName;
  final String phoneNumber;
  final String aboutMe;

  const ChatUser(
      {required this.id,
      required this.photoUrl,
      required this.displayName,
      required this.phoneNumber,
      required this.aboutMe});
```

Sl. 4.1. Model klasa korisnika

Kako bi se korisnik mogao uspješno dokazati identitet potreban je model koji će sadržavati atribute korisnika. Unutar *ChatUser* klase nalaze se dvije glavne metode :

- *toJson* koja pretvara atribute korisnika u JSON format.
- *factory* metoda koja čita podatke sa snimka kojeg dobijemo od strane Firebase baze podataka.

Napravljena je *AuthProvider* klasa koja rukuje Googleovim metodama prijave i odjave. Također služi za provjeru je li korisnik prijavljen ili ne uz pomoć metoda *handleGoogleSignIn* i *isLoggedIn*.

```dart
Future<bool> handleGoogleSignIn() async {
  _status = Status.authenticating;
  notifyListeners();

  GoogleSignInAccount? googleUser = await googleSignIn.signIn();
  if (googleUser != null) {···
  } else {
    _status = Status.authenticateCanceled;
    notifyListeners();
    return false;
  }
}

Future<bool> isLoggedIn() async {
  bool isLoggedIn = await googleSignIn.isSignedIn();
  if (isLoggedIn &&
      prefs.getString(FirestoreConstants.id)?.isNotEmpty == true) {
    return true;
  } else {
    return false;
  }
}
```

Sl. 4.2. Metode za rukovanje Googleovim metodama prijave i odjave

```dart
class _LoginPageState extends State<LoginPage> {
  @override
  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);

    switch (authProvider.status) {
      case Status.authenticateError:
        Fluttertoast.showToast(msg: 'Sign in failed');
        break;
      case Status.authenticateCanceled:
        Fluttertoast.showToast(msg: 'Sign in cancelled');
        break;
      case Status.authenticated:
        Fluttertoast.showToast(msg: 'Sign in successful');
        break;
      default:
        break;
    }
```

Sl. 4.3. Provjera identiteta korisnika

Za provjeru statusa provjere identiteta korisnika korištena je klasa *AuthProvider*. Napravljena je instanca *AuthProvider* klase, a sama provjera se odvija u zaslonu za prijavu korisnika. Nakon što se korisnik prijavi pojavi se *toast* poruka sa statusom provjere identiteta.

Kada korisnik otvori aplikaciju otvara se pozdravni zaslon koji korisnika vodi prema početnom zaslonu ili na zaslon za prijavu ovisno o tome je li korisnik već prijavljen ili nije.

```
void checkSignedIn() async {
  AuthProvider authProvider = context.read<AuthProvider>();
  bool isLoggedIn = await authProvider.isLoggedIn();
  if (isLoggedIn) {
    // ignore: use_build_context_synchronously
    Navigator.pushReplacement(
        context, MaterialPageRoute(builder: (context) => const HomePage()));
    return;
  }
  // ignore: use_build_context_synchronously
  Navigator.pushReplacement(
      context, MaterialPageRoute(builder: (context) => const LoginPage()));
}
```

Sl. 4.4. Navigacija prema različitim zaslonima

4.2. Izgradnja početnog zaslona

Kako bi se izgradio početni zaslon napravljena je *HomeProvider* klasa koja služi kao pomagač. Sadrži dvije bitne metode a to su *updateFirestoreData* koja ažurira podatke na Firestore bazi podataka i *getFirestoreData* koja dohvaća snimak podataka sa Firestore baze podataka.

```
Future<void> updateFirestoreData(
    String collectionPath, String path, Map<String, dynamic> updateData) {
  return firebaseFirestore
      .collection(collectionPath)
      .doc(path)
      .update(updateData);
}

Stream<QuerySnapshot> getFirestoreData(
    String collectionPath, int limit, String? textSearch) {
  if (textSearch?.isNotEmpty == true) {
    return firebaseFirestore
        .collection(collectionPath)
        .limit(limit)
        .where(FirestoreConstants.displayName, isEqualTo: textSearch)
        .snapshots();
  } else {
    return firebaseFirestore
        .collection(collectionPath)
        .limit(limit)
        .snapshots();
  }
}
```

Sl. 4.5. Glavne metode *HomeProvider* klase

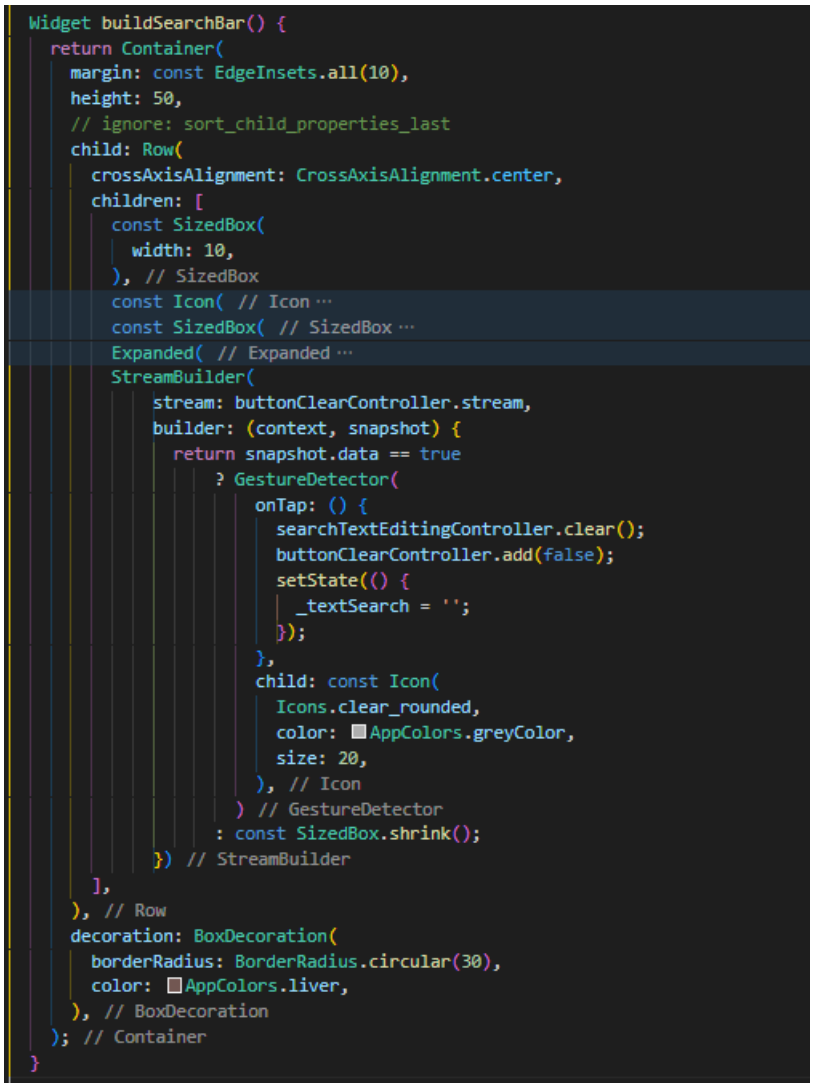
Početni zaslon sastoji se od tri dijela. Widget *AppBar* sadrži dva gumba :

- Gumb za odjavu korisnika.
- Gumb za navigaciju na zaslon profila.

Također sadrži traku za pretraživanje korisnika unutar aplikacije i *ListView* widget pomoću kojeg se prikazuju korisnici aplikacije.

```
Widget build(BuildContext context) {
  return Scaffold(
      appBar:
          AppBar(centerTitle: true, title: const Text('WidgeChat'), actions: [
        IconButton(
            onPressed: () => googleSignOut(), icon: const Icon(Icons.logout)), // IconButton
        IconButton(
            onPressed: () {
              Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) => const ProfilePage())); // MaterialPageRoute
            },
            icon: const Icon(Icons.person)), // IconButton
```

Sl. 4.6. *AppBar* widget početnog zaslona



```
Widget buildSearchBar() {
  return Container(
    margin: const EdgeInsets.all(10),
    height: 50,
    // ignore: sort_child_properties_last
    child: Row(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        const SizedBox(
          width: 10,
        ), // SizedBox
        const Icon( // Icon ···
        const SizedBox( // SizedBox ···
        Expanded( // Expanded ···
        StreamBuilder(
            stream: buttonClearController.stream,
            builder: (context, snapshot) {
              return snapshot.data == true
                  ? GestureDetector(
                      onTap: () {
                        searchTextEditingController.clear();
                        buttonClearController.add(false);
                        setState(() {
                          _textSearch = '';
                        });
                      },
                      child: const Icon(
                        Icons.clear_rounded,
                        color: AppColors.greyColor,
                        size: 20,
                      ), // Icon
                    ) // GestureDetector
                  : const SizedBox.shrink();
            }) // StreamBuilder
      ],
    ), // Row
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(30),
      color: AppColors.liver,
    ), // BoxDecoration
  ); // Container
}
```

Sl. 4.7. Izgradnja widgeta trake za pretraživanje

```
Widget buildItem(BuildContext context, DocumentSnapshot? documentSnapshot) {
  final firebaseAuth = FirebaseAuth.instance;
  if (documentSnapshot != null) {
    ChatUser userChat = ChatUser.fromDocument(documentSnapshot);
    if (userChat.id == currentUserId) {···
    } else {
      return TextButton(
        onPressed: () {···
        child: ListTile(
          leading: userChat.photoUrl.isNotEmpty
              ? ClipRRect(
                  borderRadius: BorderRadius.circular(30),
                  child: Image.network( // Image.network ···
                ) // ClipRRect
              : const Icon(
                  Icons.account_circle,
                  size: 50,
                ), // Icon
          title: Text(
            userChat.displayName,
            style: const TextStyle(color: Colors.black),
          ), // Text
        ), // ListTile
      ); // TextButton
    }
  } else {
    return const SizedBox.shrink();
  }
}
```

Sl. 4.8. Izgradnja widgeta za prikaz korisnika aplikacije

4.3. Izgradnja profilnog zaslona

Kao i kod početnog zaslona za profilni zaslon napravljena je *ProfileProvider* klasa koja sadrži dvije bitne metode. *UploadImageFile* metoda koja postavlja, odnosno, ažurira sliku profila korisnika *UpdateFirestoreData* metoda ažurira podatke korisnika.

```
UploadTask uploadImageFile(File image, String fileName) {
  Reference reference = firebaseStorage.ref().child(fileName);
  UploadTask uploadTask = reference.putFile(image);
  return uploadTask;
}

Future<void> updateFirestoreData(String collectionPath, String path,
    Map<String, dynamic> dataUpdateNeeded) {
  return firebaseFirestore
      .collection(collectionPath)
      .doc(path)
      .update(dataUpdateNeeded);
}
```

Sl. 4.9. Glavne metode *ProfileProvider* klase

Kod *ProfilePage* widgeta dvije najbitnije metode su *getImage* koja pomoću *ImagePicker* biblioteke postavlja odnosno ažurira profilnu sliku korisnika i *uploadFile* koja sprema URL podatke slike u Firestore bazu podataka.

```
Future getImage() async {
  ImagePicker imagePicker = ImagePicker();
  // PickedFile is not supported
  // Now use XFile?
  XFile? pickedFile = await imagePicker
      .pickImage(source: ImageSource.gallery)
      .catchError((onError) {
    Fluttertoast.showToast(msg: onError.toString());
  });
  File? image;
  if (pickedFile != null) {
    image = File(pickedFile.path);
  }
  if (image != null) {
    setState(() {
      avatarImageFile = image;
      isLoading = true;
    });
    uploadFile();
  }
}

Future uploadFile() async {
  String fileName = id;
  UploadTask uploadTask =
      profileProvider.uploadImageFile(avatarImageFile!, fileName);
  try {
    TaskSnapshot snapshot = await uploadTask;
    photoUrl = await snapshot.ref.getDownloadURL();
    ChatUser updateInfo = ChatUser(
        id: id,
        photoUrl: photoUrl,
        displayName: displayName,
        phoneNumber: phoneNumber,
        aboutMe: aboutMe);
    profileProvider
        .updateFirestoreData(
            FirestoreConstants.pathUserCollection, id, updateInfo.toJson())
        .then((value) async {
      await profileProvider.setPrefs(FirestoreConstants.photoUrl, photoUrl);
      setState(() {
        isLoading = false;
      });
    });
  } on FirebaseException catch (e) {
    setState(() {
      isLoading = false;
    });
    Fluttertoast.showToast(msg: e.toString());
  }
}
```

Sl. 4.10. Glavne metode *ProfilePage* widgeta

4.4.Izgradnja zaslona za razgovor

Kao i u prethodnim slučajevima kao pomoćna klasa napravljena je *ChatProvider* klasa koja sadrži četiri metode potrebne za slanje i primanje poruka.

Za učitavanje slika koje korisnici šalju jedan drugome koristi se *uploadImageFile* metoda.

Za ažuriranje informacija između korisnika koji razgovaraju koristi se *updateFirestoreData*

```
void sendChatMessage(String content, int type, String groupChatId,
    String currentUserId, String peerId) {
  final String firestoreConstant;
  if (type != 3) {
    firestoreConstant = FirestoreConstants.pathMessageCollection;
  } else {
    firestoreConstant = FirestoreConstants.pathFilesCollection;
  }
  DocumentReference documentReference = firebaseFirestore
      .collection(firestoreConstant)
      .doc(groupChatId)
      .collection(groupChatId)
      .doc(DateTime.now().toString());
  ChatMessages chatMessages = ChatMessages(
      idFrom: currentUserId,
      idTo: peerId,
      timestamp: DateTime.now().millisecondsSinceEpoch.toString(),
      content: content,
      type: type); // ChatMessages

  FirebaseFirestore.instance.runTransaction((transaction) async {
    transaction.set(documentReference, chatMessages.toJson());
  });
}
```

Sl. 4.11. Metoda *sendChatMessage*

metoda. Metoda *getChatMessages* koristi se da bi se dobio tok poruka iz Firestore baze podataka dok korisnici međusobno razgovaraju. Nakon što se poruka pošalje korisniku potrebno ju je spremiti u Firestore bazu podataka za što se koristi *sendChatMessage* metoda.

Kada korisnik pošalje poruku *sendChatMessage* metoda sprema tu poruku u Firestore bazu podataka. Ovisno o tipu poruke koja je poslana sprema se u previđenu kolekciju podataka. Nakon što se šalje poruka kreira se *FirebaseFirestore* instanca i koristi *runTransaction* metoda koja ažurira vrijednost polja na temelju njegove trenutne vrijednosti ili na temelju vrijednosti nekog drugog polja.

Što se tiče izgradnje widgeta zaslona za razgovor napravljene su dvije metode :

- *buildListMessage*.
- *buildMessageInput*.

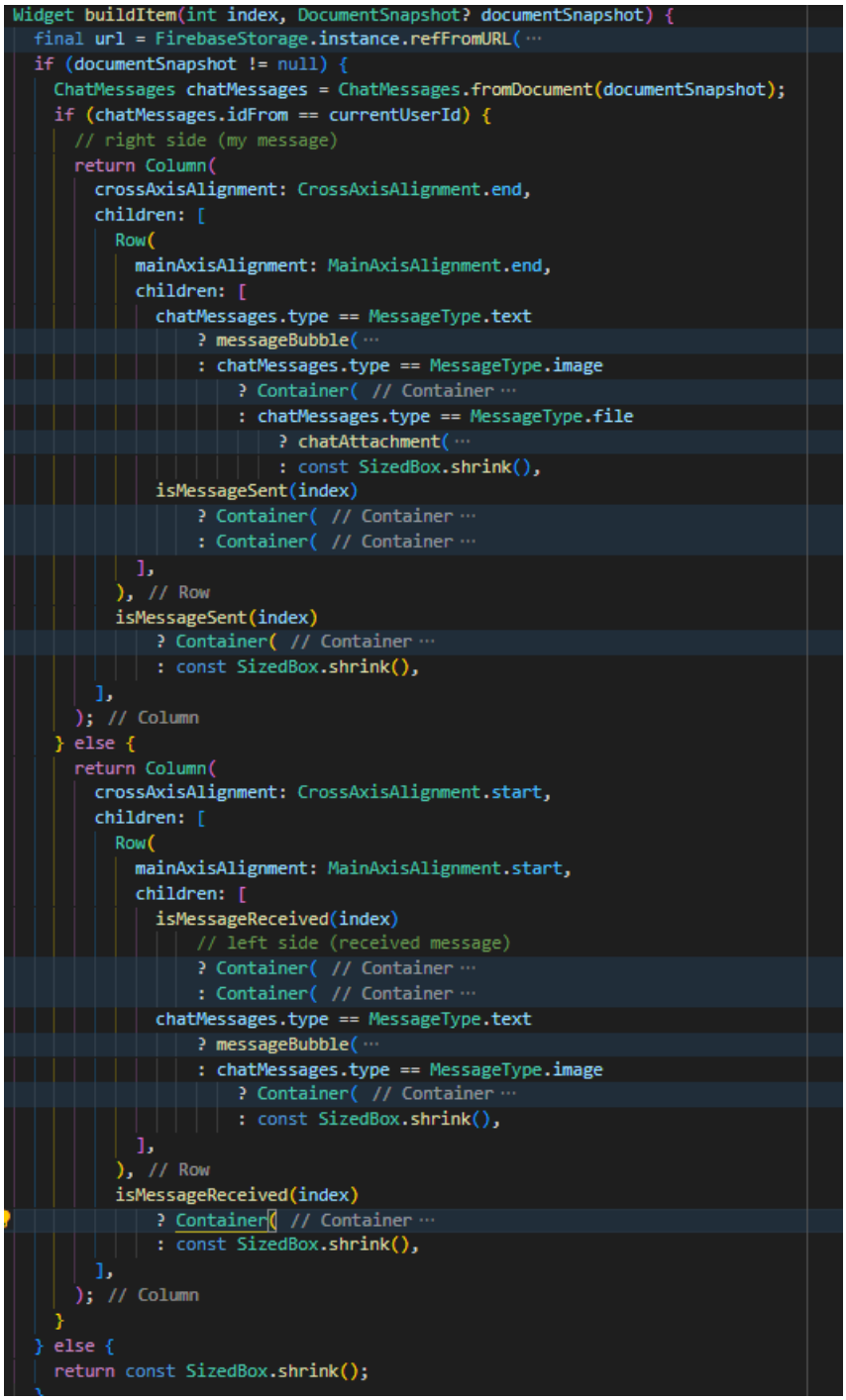
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      centerTitle: true,
      title: Text('Chatting with ${widget.peerNickname}'.trim()),
    ), // AppBar
    body: SafeArea(
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 8),
        child: Column(
          children: [
            buildListMessage(),
            buildMessageInput(),
          ],
        ), // Column
      ), // Padding
    ), // SafeArea
  ); // Scaffold
}
```

Sl. 4.12. Isječak kôda koji prikazuje izgradnju zaslona za razgovor korisnika

Metoda *buildListMessage* brine se o izgradnji widgeta za prikaz poruka između dva korisnika dok *buildMessageInput* osigurava mogućnost slanja tekstualnih poruka, slika i datoteka.

Metoda *buildListMessage* u sebi sadrži metodu *buildItem* koja se brine o izgradnji svake poruke zasebno.

Kako bi se svaka poruka mogla zasebno izgraditi potrebno je dohvatiti snimak dokumenta sa Firebase baze podataka. Zatim se provjerava gradi li se poruka od strane korisnika koji šalje ili koji prima poruku. Sljedeći korak je provjera tipa poruke koja je poslana. Ukoliko je tip poruke tekstualna poruka izgradit će se widget pomoću metode *messageBubble* koja će proizvest klasični izgled poruke sa tekstualnim sadržajem. Ako je tip poruke slika izgradit će se widget koji vraća spremnik sa poslanom slikom. Nadalje, *isMessageSent* metoda zatim provjerava je li poruka poslana te ispisuje datum i vrijeme slanja poruke.



Sl. 4.13. Metoda *buildItem*

Metoda *buildMessageInput* vraća *SizedBox* widget u kojem se nalaze tri gumba i *TextField* widget. Prvi gumb služi za odabiranje slike koju korisnik želi poslati, drugi za odabiranje datoteke, treći

služi kao dodatan gumb za slanje poruke. Pritiskom na gumb za odabiranje slike aktivira se asinkrona metoda *getImage* koja koristi *ImagePicker* biblioteku za odabiranje slika iz galerija mobilnog uređaja. Pritiskom na gumb za odabiranje datoteke aktivira se asinkrona metoda *getFile* koja koristi

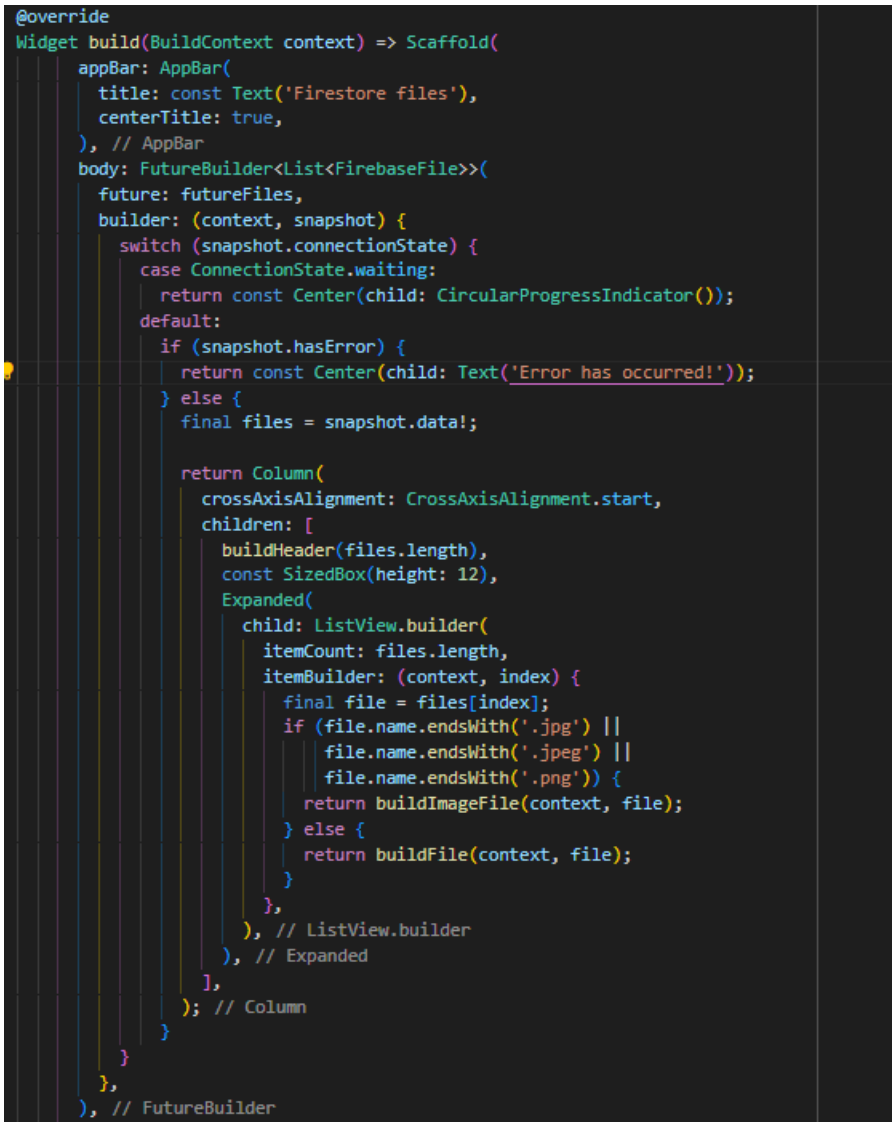
```
Widget buildMessageInput() {
  return SizedBox(
    width: double.infinity,
    height: 50,
    child: Row(
      children: [
        Container( // Container ···
        Container( // Container ···
        Flexible(
            child: TextField(
          focusNode: focusNode,
          textInputAction: TextInputAction.send,
          keyboardType: TextInputType.text,
          textCapitalization: TextCapitalization.sentences,
          controller: textEditingController,
          decoration:
              kTextInputDecoration.copyWith(hintText: 'write here...'),
          onSubmitted: (value) {
            onSendMessage(textEditingController.text, MessageType.text);
          },
        )), // TextField // Flexible
        Container( // Container ···
      ],
    ), // Row
  ); // SizedBox
}
```

Sl. 4.14. Metoda *buildMessageInput*

4.5. Izgradnja zaslona za preuzimanje datoteka

Zaslon za preuzimanje datoteka sastoji se od :

- *AppBar* widgeta.
- Widgeta *buildHeader* na kojem je prikazana ikona i broj datoteka koje su razmijenjene u razgovoru između korisnika.
- *ListView* widgeta koji prikazuje listu svih datoteka koje se mogu preuzeti.



```dart
@override
Widget build(BuildContext context) => Scaffold(
      appBar: AppBar(
        title: const Text('Firestore files'),
        centerTitle: true,
      ), // AppBar
      body: FutureBuilder<List<FirebaseFile>>(
        future: futureFiles,
        builder: (context, snapshot) {
          switch (snapshot.connectionState) {
            case ConnectionState.waiting:
              return const Center(child: CircularProgressIndicator());
            default:
              if (snapshot.hasError) {
                return const Center(child: Text('Error has occurred!'));
              } else {
                final files = snapshot.data!;

                return Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    buildHeader(files.length),
                    const SizedBox(height: 12),
                    Expanded(
                      child: ListView.builder(
                        itemCount: files.length,
                        itemBuilder: (context, index) {
                          final file = files[index];
                          if (file.name.endsWith('.jpg') ||
                              file.name.endsWith('.jpeg') ||
                              file.name.endsWith('.png')) {
                            return buildImageFile(context, file);
                          } else {
                            return buildFile(context, file);
                          }
                        },
                      ), // ListView.builder
                    ), // Expanded
                  ],
                ); // Column
              }
          }
        },
      ), // FutureBuilder
```

Sl. 4.15. Izgradnja zaslona za preuzimanje datoteka

Kada korisnik pošalje sliku ili datoteku drugome korisniku te datoteke se spremaju na Firebase bazi podataka. Sve što je poslano između dva korisnika može se vidjeti ovom zaslonu. Kao pomoćni zaslon napravljen je i *ImagePage* zaslon. Služi za pregled datoteka odnosno slika koje završavaju sa nastavkom *.jpg, .jpeg* ili *.png*. Navigacija do *ImagePage* zaslona odvija se u *buildImageFile* widgetu.

```
Widget buildImageFile(BuildContext context, FirebaseFile file) => ListTile(
    leading: ClipOval(
      child: Image.network(
        file.url,
        width: 52,
        height: 52,
        fit: BoxFit.cover,
      ), // Image.network
    ), // ClipOval
    title: Text(
      file.name,
      style: const TextStyle(
        fontWeight: FontWeight.bold,
        decoration: TextDecoration.underline,
        color: Colors.black,
      ), // TextStyle
    ), // Text
    onTap: () => Navigator.of(context).push(MaterialPageRoute(
      builder: (context) => ImagePage(file: file),
    )), // MaterialPageRoute
  ); // ListTile
```

Sl. 4.16. Isječak kôda koji predstavlja *buildImageFile* widget

U *DownloadPage* zaslonu ako je datoteka slika korisniku je omogućen pregled slike u ovalnom obliku pomoću *ClipOval* widgeta. Kada korisnik pritisne na sliku navigacija ga vodi na *ImagePage* zaslon gdje se odabrana slika može pregledati u punom obliku.

```dart
@override
Widget build(BuildContext context) {
  final isImage = ['.jpeg', '.jpg', '.png'].any(file.name.contains);

  return Scaffold(
    appBar: AppBar(
      title: Text(file.name),
      centerTitle: true,
      actions: [
        IconButton(
          icon: const Icon(Icons.file_download),
          onPressed: () async {
            await ChatProvider.downloadFile(file.ref);

            final snackBar = SnackBar(
              content: Text('Downloaded ${file.name}'),
            ); // SnackBar
            // ignore: use_build_context_synchronously
            ScaffoldMessenger.of(context).showSnackBar(snackBar);
          },
        ), // IconButton
        const SizedBox(width: 12),
      ],
    ), // AppBar
    body: isImage
        ? Image.network(
            file.url,
            height: double.infinity,
            fit: BoxFit.cover,
          ) // Image.network
        : const Center(
            child: Text(
              'Cannot be displayed',
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ), // Text
          ), // Center
  ); // Scaffold
}
```

Sl. 4.17. Izgradnja *ImagePage* zaslona za pregled slika

ImagePage zaslon sadrži *AppBar* widgeta, a u *AppBar* widgetu se nalazi I*conButton* widget koji služi za preuzimanje slike. Kada korisnik preuzme sliku prikazat će se *toast* poruka koja obavještava da se dokument uspješno preuzeo. Na ovom zaslonu slika se prikazuje pomoću *Image.network()* metode koja stvara widget koji prikazuje slikovni tok podataka dobiven s mreže.

5. IZGLED APLIKACIJE

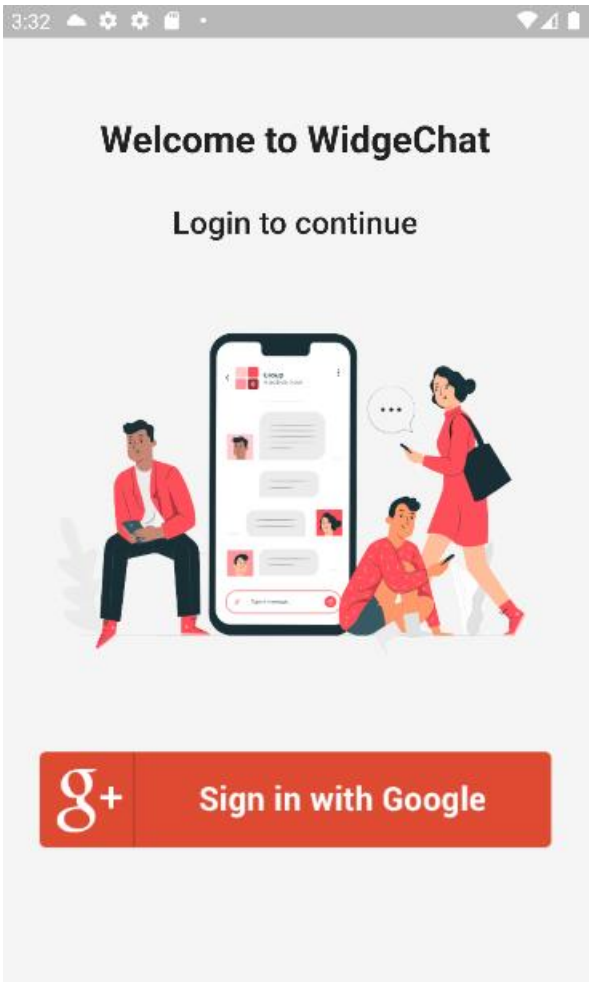
Sučelje mobilne aplikacije sastoji se od sedam zaslona a to su :

- Prijava.
- Prikaz poruka (glavni zaslon).
- Prikaz profila (početna stranica).
- Zaslona za razgovor (specifični kanal dva korisnika).
- Pozdravni zaslon.
- Zaslona za preuzimanje datoteka.
- Zaslon za pregled slika.

5.1. Zaslon za prijavu

Kada korisnik otvori aplikaciju prikazuje se početni zaslon koji se sastoji od tekstualnog widgeta i slike u sredini.

Zaslon za prijavu sastoji se od dva tekstualna widgeta, slike i gumba za prijavu. Za provjeru identiteta korisnika koristi se instanca *AuthProvider* klase. Ukoliko je korisnik uspješno dokazao identitet aplikacija će korisnika preusmjeriti na početni zaslon sa *toast* porukom da je uspješno prijavljen (Slika 5.1.).



Sl. 5.1. Prikaz početnog zaslona za prijavu korisnika

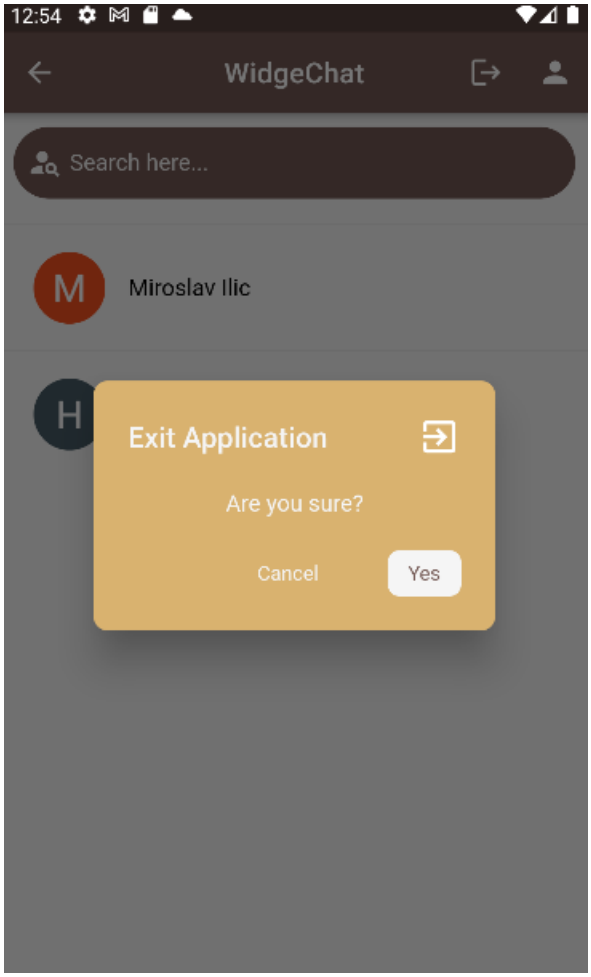
5.2. Početni zaslon

Početni zaslon aplikacije sastoji se od *Scaffold* widgeta koji sadrži *AppBar* widget u kojem se nalaze tri gumba :

- Izlazak iz aplikacije.
- Preusmjeravanje na profilni zaslon.
- Za odjavu korisnika.

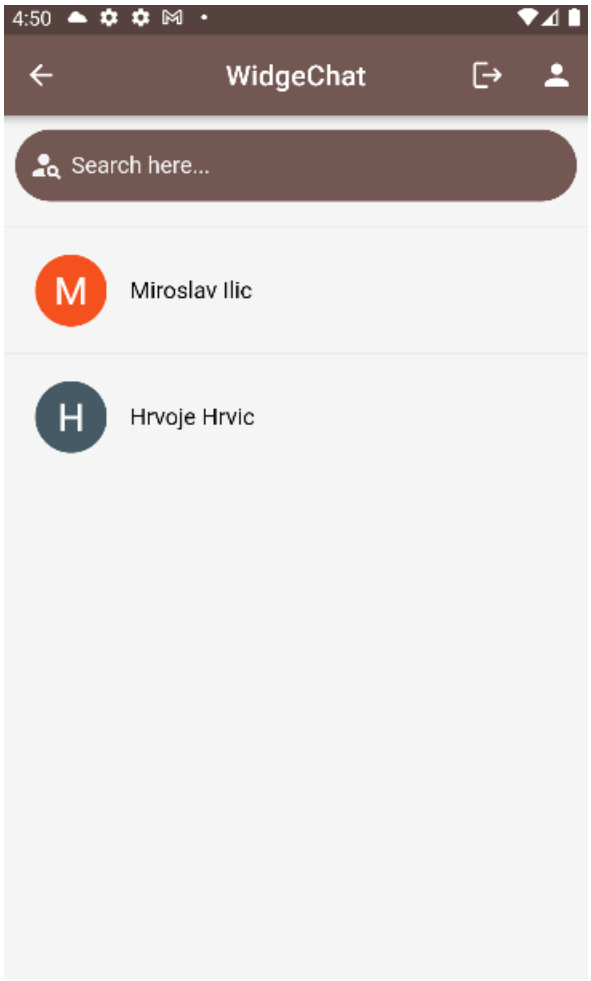
Također, sadrži trake za pretraživanje korisnika *ListTile* widget i profilnu sliku od Gmail korisničkog računa.

Pritiskom na gumb za izlazak iz aplikacije otvara se dijalog koji omogućuje korisniku izlazak iz aplikacije (Slika 5.2.5.2.).



Sl. 5.2. Otvaranje dijaloga nakon pritiska na gumb za izlazak

Traka za pretraživanje korisnika služi za pretragu korisnika koji su prijavljeni u aplikaciji. Aplikacija je pojednostavljena na način da su svi korisnici koji su prijavljeni vidljivi drugim korisnicima. Obzirom da može doći do velikog broj korisnika na ovaj način se olakšava korištenje aplikacije (Slika 5.3.).



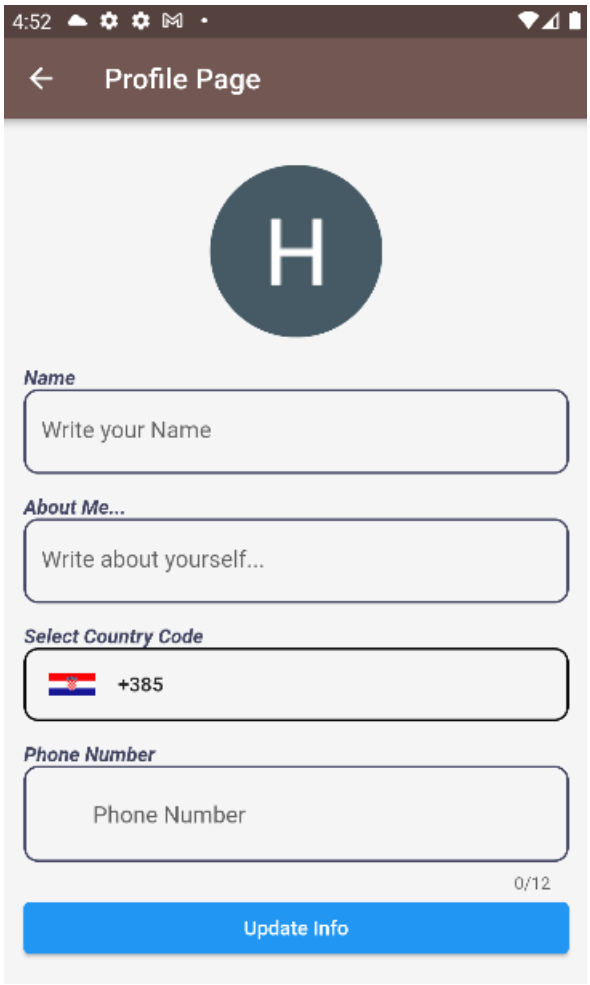
Sl. 5.3. Prikaz početnog zaslona korisnika

5.3. Zaslon za prikaz profila

Zaslon za prikaz profila sastoji se od:

- Tri tekstualna widgeta.
- Profilne slike.
- Gumb widgeta za ažuriranje podataka.

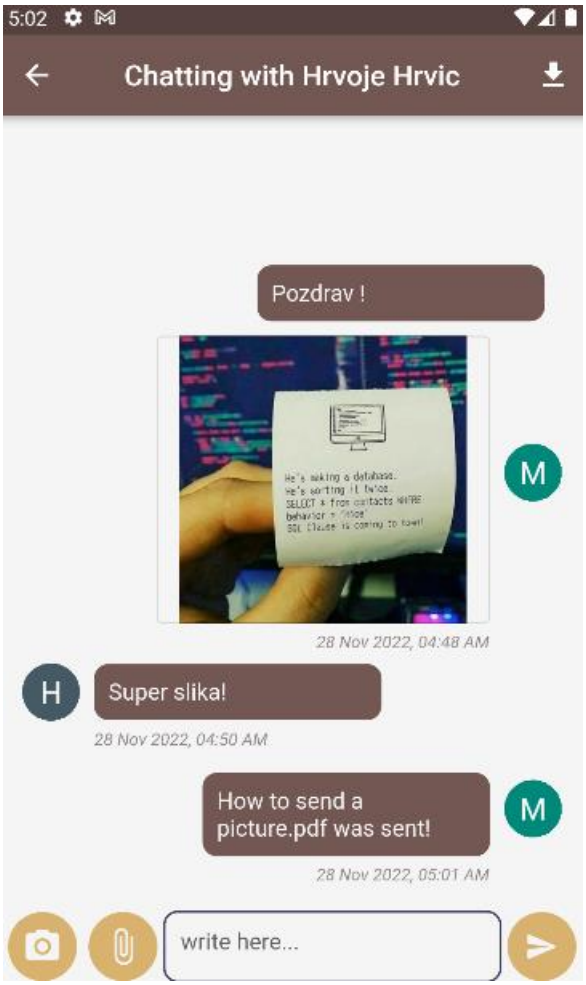
Na ovom zaslonu moguće je ažurirati podatke korisnika u Firebase bazi podataka. Korisnici mogu ažurirati svoje korisničko ime, napisati nešto o sebi i dodati svoj broj mobilnog telefona.



Sl. 5.4. Prikaz zaslona profila korisnika

5.4. Zaslon za razgovor između korisnika

Zaslon za razgovor između korisnika sličan je kao i u većini aplikacija koje služe za dopisivanje. Sastoji se od *AppBar* widgeta, dijela za prikaz poruka između dva korisnika, tekstualnog polja za upis poruka i dva gumba za slanje slika i datoteka. Moguće je poslati različite vrste datoteka koje se spremaju u Firebase bazu podataka. Kada se pošalje datoteka korisnik je obavješten sa porukom u obliku imena poslane datoteke i „was sent!“ nastavka. Za preuzimanje poslane datoteke potrebno je otići na gumb za skidanje datoteka koji se nalazi na *AppBar* widgetu. Gumb će korisnika odvesti na novi zaslon gdje su vidljive sve poslane datoteke u trenutnom razgovoru (Slika 5.5.).

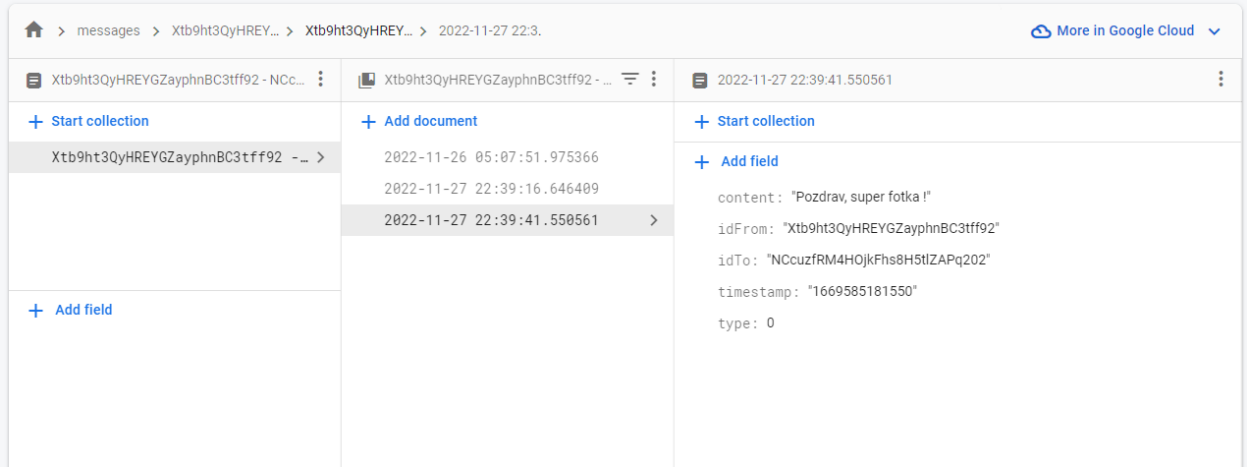


Sl. 5.5. Prikaz zaslona za razgovor između korisnika

Kada korisnik pošalje poruku na Firestore bazi podataka stvara se novi dokument sa nazivom ID razgovora. ID razgovora je uvijek ID pošiljatelja – ID primatelja poruke. Unutar tog dokumenta može se pronaći kolekcija dokumenata u kojima je sadržaj poruke korisnika kao i neke druge podatke poput:

- ID pošiljatelja.
- ID primatelja.
- Vremensku oznaku u UNIX formatu.
- Tip poruke.

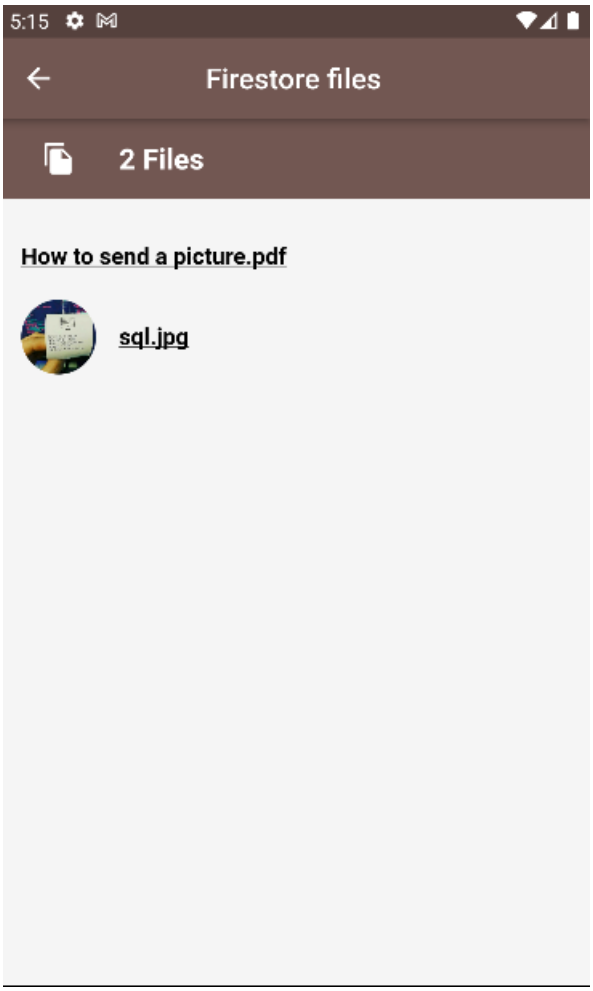
Opisano je prikazano na slici 5.6.



Sl. 5.6. Prikaz poruke spremljene u Firestore bazi podataka

5.5. Zaslon za preuzimanje datoteka

Da bi korisnik mogao doći do ovog zaslona potrebno je pritisnuti gumb za preuzimanje datoteka koji se nalazi u *AppBar* widgetu na zaslonu za razgovor između korisnika. Pritiskom na naziv datoteke započinje se preuzimanje. Kada se datoteka preuzme prikaže se *toast* poruka koja obavještava korisnika o uspješnom preuzimanju datoteke (Slika 5.7.). Za datoteke koje su ekstenzija *.jpg, jpeg i .png* moguć je pregled u punom obliku na pomoćnom *ImagePage* zaslonu (Slika 5.8.). Kada korisnik pritisne na sliku u zaslonu za preuzimanje datoteka preusmjerava se na *ImagePage* zaslon. Moguće je preuzeti sliku na mobilni uređaj pomoću gumba koji se nalazi u *AppBar* widgetu *ImagePage* zaslona.



Sl. 5.7. Prikaz zaslona za preuzimanje datoteka

Sl. 5.8. Prikaz zaslona za preuzimanje datoteka

6. ZAKLJUČAK

Flutter je relativno nova tehnologija koja je dostupna tek nekoliko godina. Iako izvorne imaju svoje očite prednosti kao što su glatkoća i brzina koju je teško sa višeplatformskim tehnologijama postići višeplatformska okruženja pronalaze svoju upotrebu u raznim sektorima razvoja mobilnih aplikacija.

Pomoću Flutter programskog okruženja realizirana je mobilna aplikacija za dopisivanje. Korisnicima je omogućeno slanje poruka tekstualnog sadržaja, slika i datoteka. Koristeći Firebase bazu podataka izgradnja poslužiteljske aplikacije je znatno olakšana. Opisana je komunikacija poslužitelja i klijenta kroz metode pomoćnih klasa kao i metoda za izgradnju widgeta, odnosno, temeljnih blokova zaslona aplikacije. Objašnjena je provjera identiteta korisnika u aplikaciji. Izgled i funkcionalnost zaslona aplikacije su prikazani i objašnjeni slikama. Testiranjem aplikacije utvrđena je njena funkcionalnost.

Proces prilagodne aplikacije za svaku platformu je kompleksan jer je potrebno graditi grafičko sučelje Widget elementima odnosno elementima grafičkog sučelja koji se prilagođavaju svakoj platformi. Uz relativno jednostavno poznavanje razvojnog okvira Flutter i programskog jezika Dart, kroz praćenje dokumentacije i jednostavnih smjernica, Flutter razvojno okruženje pruža programerima odličnu priliku za izradu aplikacija koje se mogu usporediti aplikacijama izrađenim u izvornim tehnologijama.

LITERATURA

[1] Anoninmno, »What is a native mobile app development,« [Mrežno]. Dostupno na: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development-. [Pokušaj pristupa 12. Studeni 2022.].

[2] Anonimmo, »Android studio guide,« [Mrežno]. Dostupno na: https://developer.android.com/

[Pokušaj pristupa 12. Studeni 2022.].

[3] Anonimno, »Android studio Dolphin,« [Mrežno]. Dostupno na: https://developer.android.com/studio/releases [Pokušaj pristupa 13. Studeni 2022.].

[4] Anonimno, »Xcode,« [Mrežno]. Dostupno na: https://developer.apple.com/documentation/xcode [Pokušaj pristupa 13. Studeni 2022.].

[5] Anonimno, »Flutter FAQ,« [Mrežno]. Dostupno na: https://docs.flutter.dev/resources/faq [Pokušaj pristupa 14. Studeni 2022.].

[6] Anonimno, »Hot reload,« [Mrežno]. Dostupno na: https://docs.flutter.dev/development/tools/hot-reload [Pokušaj pristupa 14. Studeni 2022.].

[7] Anonimno, »Flutter architectural overview,« [Mrežno]. Dostupno na: https://docs.flutter.dev/resources/architectural-overview [Pokušaj pristupa 14. Studeni 2022.].

[8] Anonimno, »A tour of the Dart language,« [Mrežno]. Dostupno na: https://dart.dev/guides/language/language-tour [Pokušaj pristupa 14. Studeni 2022.].

[9] T.Goel, »Lifecycle of Flutter App,« [Mrežno]. Dostupno na: https://mobikul.com/lifecycle-of-a-flutter-app/ [Pokušaj pristupa 17. Studeni 2022.].

[10] Anonimno, »WhatsApp overview,« [Mrežno]. Dostupno na : https://developers.facebook.com/docs/whatsapp/ [Pokušaj pristupa 18. Studeni 2022.].

[11] Anonimno, »Number of unique Viber users Ids from June 2011 to March 2020,« [Mrežno]. Dostupno na : https://www.statista.com/statistics/316414/viber-messenger-registered-users/ [Pokušaj pristupa 18. Studeni 2022.].

[12] Anonimno, »BlablaCar,« [Mrežno]. Dostupno na : https://support.blablacar.com/hc/hr/ [Pokušaj pristupa 18. Studeni 2022.].

[13] S. Melendez, »Sometimes you're just one hop from something huge,« [Mrežno]. Dostupno na : https://www.fastcompany.com/3031109/sometimes-youre-just-one-hop-from-something-huge [Pokušaj pristupa 19. Studeni 2022.].

SAŽETAK

Tema ovog diplomskog rada je mobilna chat aplikacija, a cilj rada je omogućiti korisnicima razmjenu poruka tekstualnog sadržaja, slika i datoteka. Za potrebe rada napravljena je klijentska aplikacija uz Firestore bazu podataka koja služi kao poslužiteljski dio. Klijentski dio aplikacije odrađen je uz pomoć Flutter razvojnog okruženja.

Ključne riječi: Dopisivanje, Firebase, Flutter, Višeplatformski

ABSTRACT

MOBILE CHAT APPLICATION

The topic of this thesis is a mobile chat application, and the goal of this work is to enable users to exchange messages of textual content, as well as images and files. For work purposes, a client application was created with Firestore database serving as backend. The client part of the application was developed with the help of Flutter development environment.

Keywords : Chat, Cross-platform, Flutter, Firebase