Šolski center Novo mesto

Srednja elektro-računalniška šola in tehnična gimnazija

Šegova ulica 112

8000 Novo mesto

APLIKACIJA ZA POGOVOR PREKO SPLETA

(Izdelek oziroma storitev in zagovor)

Avtor: Pavel Krnc R4B

Mentor: mag. dru. inf. Simon Vovko

Pečice, maj 2024

Zahvala

Zahvaljujem se mojemu mentorju mag. dru. inf. Simonu Vovku za nasvete pri opravljanju zaključnega izdelka poklicne mature. Zahvaljujem se tudi vsem sošolcem, ki ste mi pomagali iskati napake v aplikaciji.

Povzetek

Za izdelek sem se odločil narediti android aplikacijo. Aplikacija omogoča pogovor uporabnikov preko spleta ter igranje preprostih iger z ostalimi uporabniki. Za ta projekt sem se odločil predvsem zato, da bi razširil moje znanje o izdelovanju mobilnih aplikacij in programiranju na splošno. Aplikacijo sem izdelal z razvojnim okoljem Flutter, za potrebne spletne storitve sem pa uporabil spletno platformo Firebase.

Med programiranjem aplikacije se mi je svet programiranja predstavil na popolnoma nov način, in mi s tem omogočil boljše razumevane računalništva.

Ključne besede: Flutter, Firebase, aplikacija, Dart, gradnik

KAZALO VSEBINE

KAZALO SLIK

KAZALO SHEM

1 Uvod

1.1 Namen projekta

Osrednji cilj tega projekta je bil pridobiti praktične izkušnje in znanje na področju razvoja mobilnih aplikacij. Z izbiro razvojnega okolja Flutter sem si zagotovil možnost učenja sodobnih tehnologij, ki so ključne za izdelavo kakovostnih aplikacij na različnih platformah.

Projekt, ki sem si ga zadal se je izkazal za velik izziv vendar sem ga kljub temu uspešno dokončal in s tem osvojil znanje in praktične izkušnje na področju razvijanja mobilnih aplikacij.

1.2 O področju

Razvoj mobilnih aplikacij je področje ki se nenehno spreminja in prilagaja novim tehnološkim trendom zato se morajo razvijalci mobilnih aplikacij soočati z različnimi izzivi. Da bi pa te izzive poenostavil sem popa pri tem projektu uporabil Flutter in Firebase, ki omogočata moderne in dinamične rešitve.

Flutter je odprtokodno programsko okolje namenjen izdelavi uporabniškega vmesnika. Razvijalcem omogoča izdelavo aplikacij za najbolj znane operacijske sisteme kot na primer Android.

Firebase je spletna platforma za razvoj aplikacij. Razvijalcem ponuja vrsto storitev v oblaku, ki jim pomagajo pri gradnji, nadzoru in upravljanju mobilnih in spletnih aplikacij.

2 Teoretični del

2.1 Flutter

Flutter je odprtokodno programsko okolje ali drugače rečeno komplet za razvoj programske opreme namenjeno razvoju aplikacij vseh vrst, ki ga je razvil Google. Razvijalcem omogoča izdelavo aplikacij za spletne strani ter operacijske sisteme Fuchsia, Android, iOS, Linux, macOS in Windows. Omogoča izdelavo modernega in odzivnega uporabniškega vmesnika, preprosto povezovanje v spletne storitve in še druge uporabne funkcije, ki Flutter postavijo med najboljše platforme za izdelovanje aplikacij.

Programsko okolje Flutter vsebuje pogon za izrisovanje (ang. rendering engine) kot tudi jezik za oblikovanje (ang. UI language), kar pomeni da ni odvisen od drugih pogonov. Flutterjev popoln nadzor nad cevovodom za upodabljanje (ang. rendering pipeline) poenostavlja podporo več platform, saj potrebuje samo platformo za izvajanje izvorne kode.

Pomembno vlogo v Flutter-ju igrajo tudi gradniki (widget). To so razredi bodisi integrirani ali narejeni s strani razvijalca, ki razširijo razred »Widget«. Gradniki lahko nekaj prikažejo na zaslonu ali pa zgolj obdelujejo podatke zato jih je potrebno smiselno vezati skupaj da se sestavijo v grafični vmesnik z delovanjem. Vsebujejo funkcijo »build« katera vsebuje kar bo gradnik izrisal na zaslonu.

Aplikacije v programskem okolju Flutter programiramo v programskem jeziku Dart:

2.1.1 Dart

Dart je sodoben programski jezik, ki ga je razvilo podjetje Google. Je objektno usmerjen, ima tipovno varnost (ang. type safety) in je optimiziran za gradnjo odzivnih aplikacij na različnih platformah. Dart je bil izbran za programski jezik Flutterja iz več razlogov:

- **Hitra izvedba:** Dart omogoča tako AOT(Ahead Of Time) kot JIT(Just In Time) izgradnjo aplikacije, kar pomeni, da lahko aplikacije tečejo zelo hitro.
- **Hot Reload:** Omogoča da se spremembe v aplikaciji pojavijo brez ponovne izgradnje aplikacije.

- **Preprostost in strukturnost:** Dart je zasnovan za enostavnost in produktivnost, kar olajša uporabo razvijalcem.

Za razvojno okolje Flutter sem se odločil predvsem zato, da bi se naučil uporabe modernih razvojnih okolji.

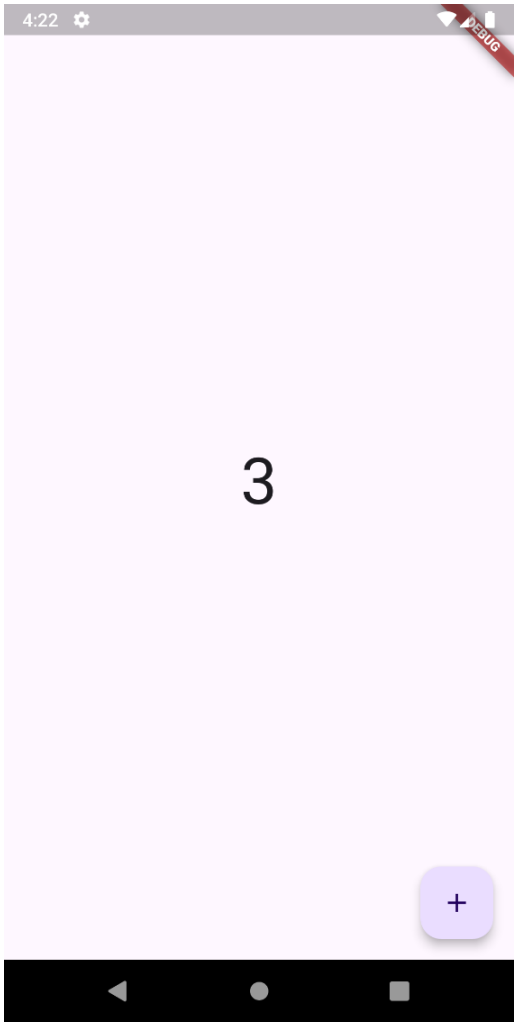
2.1.2 Primer Flutter projekta

Nov projekt v Flutter-ju naredimo tako, da v ukaznem pozivu izvedemo ukaz »flutter create [ime_projekta]« in to bo naredilo novo mapo z imenom projekta. V tej mapi bo tudi naredilo več podmap in datotek namenjenih delovanju projekta na vseh podprtih platformah. Podmape »android«, »ios«, »linux«, »macos«, »web« in »windows« so vsaka namenjene svoji platformi in v njih je potrebno samo posodobiti nekatere datoteke ob uporabi zahtevnejših funkcij na primer posamezni platformi dodati dovoljenje za uporabo interneta. Preostale podmape pa so še »build« in »lib«. V podmapi »build« se nahajajo izvedljive datoteke za vsako platformo posebej ko jih razvijalci ustvarijo. V podmapi »lib« pa se ustvari datoteka »main.dart«, ki se izvede ob zagonu programa. Podmapa »lib« se uradno tudi uporablja za shranjevanje ostalih »dart« datotek.

V datoteki »main.dart« se nahaja funkcija »main«, ki se prva izvede in zažene aplikacijo z funkcijo »runApp«, ki kot argument prejme gradnik katerega bo zgradila. V tem primeru je to »Stateful« gradnik »PrimerPage«, ki je u bistvu samo dva razreda z malo drugačno obliko. Gradnik z vgrajeno funkcijo »build« zgradi »MaterialApp« v katerem so gnezdeni ostali gradniki, ki na zaslon izrišejo številko iz spremenljivke »stevec« in gumb, ki ob kliku izvede funkcijo »klikGumba«. Funkcija »klikGumba« poveča spremenljivko »stevec« za 1 in pokliče vgrajeno funkcijo »setState«, ki posodobi vsebino aplikacije.

```dart
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() {
  runApp(const PrimerPage());
}

class PrimerPage extends StatefulWidget {
  const PrimerPage({super.key});

  @override
  _PrimerPageState createState() => _PrimerPageState();
}

class _PrimerPageState extends State<PrimerPage> {
  int stevec = 0;

  void klikGumba() {
    stevec++;
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
      body: Center(
        child: Text(
          stevec.toString(),
          style: const TextStyle(fontSize: 50),
        ), // Text
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: klikGumba,
        child: const Icon(Icons.add),
      ), // FloatingActionButton
    )); // Scaffold // MaterialApp
  }
}
```

Slika 2: Primer kode v Flutter-ju



Slika 1: Slika primera kode v Flutter-ju

2.2 Firebase

Firebase je platforma za razvoj aplikacij, ki jo je razvil Google in je namenjena poenostavitvi procesov gradnje, nadzoru in upravljanja aplikacij in spletnih strani. Je backend infrastruktura, ki razvijalcem nudi različne storitve brez da bi se morali ukvarjati z zapletenostmi strežniških storitev. Poleg tega pa je tudi spletno mesto kjer lahko razvijalci gostijo spletne strani, podatkovne baze in še nekatere ostale spletne storitve kot so:

- **Cloud Firestore:** NoSQL podatkovna baza, ki omogoča shranjevanje in delitev podatkov med uporabniki v realnem času.

- **Authentication:** Ponuja preprosto avtentikacijo uporabnikov z različnimi funkcijami.
- **Google Analytics:** Integrirana storitev, ki razvijalcem omogoča pregled uporabniškega vedenja kar omogoča izboljšanje aplikacije.
- **Cloud Messaging:** Storitev za pošiljanje obvestil uporabnikom.
- **Remote Config:** Orodje, ki razvijalcem omogoča spreminjanje vedenja in videza aplikacije brez potrebe po objavi nove različice aplikacije.

Platforma Firebase je zasnovana tako, da je enostavna za integracijo in uporabo s podporo za več platform vključno z iOS, Android, Flutter, Unity in C++. S svojim obsežnim naborom storitev za razvoj programske opreme in dokumentacijo razvijalcem omogoča hitro in učinkovito gradnjo aplikacij.

Firebase sem uporabil za shranjevanje podatkov v podatkovni bazi Firebase Firestore, varno prijavljanje v aplikacijo z orodjem Authentication, sprejemanje in prikazovanje obvestil z orodjem Firebase Messaging in shranjevanje slik in dokumentov z orodjem Firebase Storage.

2.3 Visual studio code

Visual Studio Code je zmogljiv in preprost urejevalnik izvorne kode, ki je na voljo za Windows, masOS in Linux. Urejevalnik je zasnovan za razvijalce vseh področij in ponuja preprost uporabniški vmesnik z možnostmi odpravljanja napak (ang. debugging), upravljanje datotek in direktorijev ter možnost povezave z Gitom. Visual Studio Code ima tudi možnost dodajanje različnih razširitev in orodij, ki omogočajo pridobitev podpore in dodatkov za večino programskih jezikov in platform.

Ta urejevalnik sem si izbral ker je zmogljiv ter uporabniku prijazen in mi je zato prijeten za uporabo. Z razširitvami za Flutter in Dart je pridobil veliko uporabnih funkcij, ki so mi bile v pomoč.

2.4 Android studio

Razvojno okolje Android studio je namenjeno za razvoj Android aplikacij. Android Studio je razvil Google z namenom nadomestitve Eclipse IDE. Namenjen je

zagotavljanju orodij, ki razvijalcem omogočajo učinkovito ustvarjanje, testiranje in uvajanje Android aplikacij na različne naprave, predvsem telefone in tablice.

Android Studio ponuja uporabna orodja kot so:

- **Inteligentni urejevalnik kode:** Ponuja napredno dopolnjevanje kode za Kotlin, Java in C/C++ programske jezike.
- **Device manager:** Razvijalcem omogoča testiranje aplikacij na fizičnih in virtualnih napravah. Ponuja možnost ustvaritve virtualnih naprav različnih velikosti in s katero koli različico operacijskega sistema Android.
- **Jetpack Compose orodja za oblikovanje:** Omogočajo ustvarjanje dinamičnih postavitev in pregledovanje le-teh na katerikoli velikosti zaslona ter inšpekcijo animacij.
- **Podpora GitHub:** Omogoča hitro povezavo projektov z GitHub-om kar je zelo uporabno za podjetja oziroma skupinsko razvijanje aplikacij.

2.4.1 Gradle

Gradle je sistem za avtomatizacijo gradnje aplikacije oziroma programa, ki ga uporablja Android Studio. Je ključni del razvojnega procesa, saj omogoča avtomatizacijo procesov povezanih z gradnjo in pakiranjem aplikacij. Uporablja konfiguracijske datoteke, da določi pravila za gradnjo, testiranje in pakiranje aplikacij.

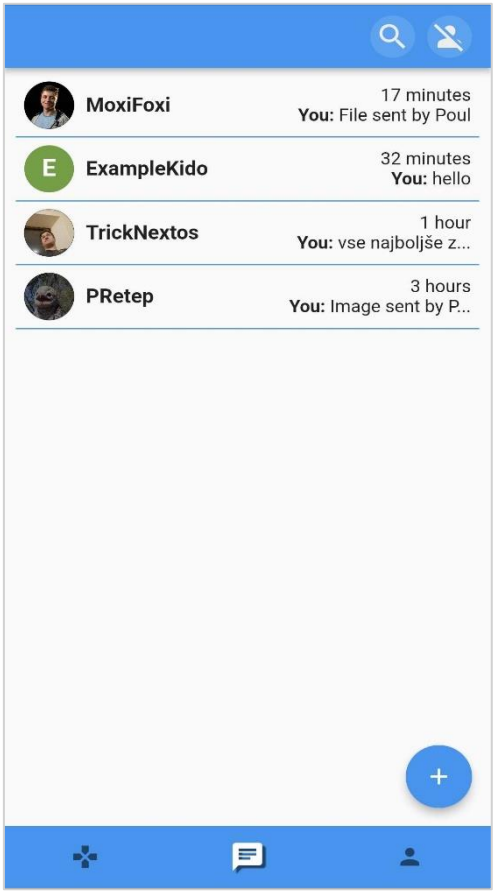
Android Studio sem uporabil ker je potrebno orodje za izgradnjo Android aplikacije in z njim je bila pridobitev virtualnih mobilnih naprav enostavnejša in hitrejša.

3 Praktični del

3.1 Končni izdelek

Končni izdelek bo aplikacija imenovana PavliText ki bo namenjena komunikaciji preko spleta. PavliText omogoča varno ustvaritev profila z email naslovom in geslom. Ko je profil ustvarjen in se oseba prijavi pa omogoča:

- Dodajanje, odstranjevanje in blokiranje uporabnikov,
- Pogovor z ostalimi uporabniki v kontaktih,
- Pošiljanje slik in datotek uporabnikom,
- Igranje preprostih iger(trenutno samo Križci in Krožci) z ostalimi uporabniki,
- Spreminjanje velikosti pisave v pogovorih,
- Možnost temnega načina,
- Odjava in ponovna prijava v aplikacijo



Slika 3: Posnetek zaslona aplikacije

3.2 Ideja

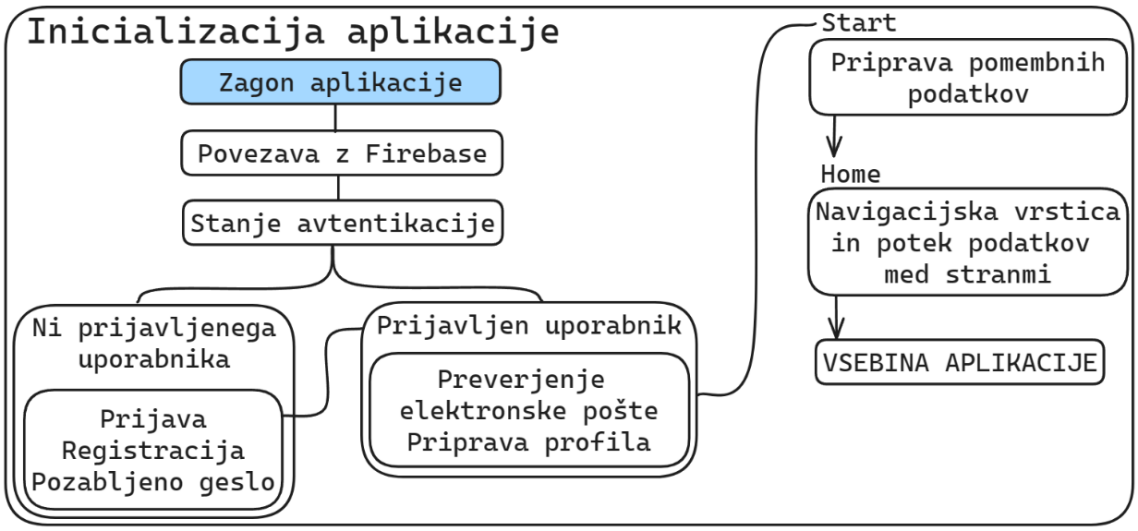
Za izdelavo aplikacije sem se odločil zato ker me je tema razvijanja aplikacij vedno zanimala in mi bo služila za pogovor s prijatelji. Aplikacija mi bo tudi služila kot dokazilo mojih sposobnosti delodajalcem, ko se bom zaposloval.

3.3 Načrtovanje aplikacije

Aplikacijo sem po svojih zmožnostih načrtoval tako, da sem ločil inicializacijo aplikacije od vsebine, kar je bil dober pristop ampak pomanjkljiv in mi je otežil dodajanje nekaterih funkcij.

3.3.1 Inicializacija aplikacije

Ob zagonu aplikacije se mora najprej vzpostaviti povezava z spletnim orodjem Firebase in hkrati sistem za prejemanje obvestil s pomočjo orodja Firebase Messaging. Nato bo aplikacija vzpostavila Stream, ki preverja ali je v aplikacijo prijavljen kakšen uporabnik. Če ne bo prijavljen noben uporabnik bo aplikacijo usmeril na del aplikacije, ki bo omogočal prijavo, registracijo in možnost za pozabljeno geslo. Ko se bo uporabnik uspešno prijavil se bo izvedel del kode ki bo preveril če je določen uporabnik že nastavil svoje uporabniško ime, profilno sliko

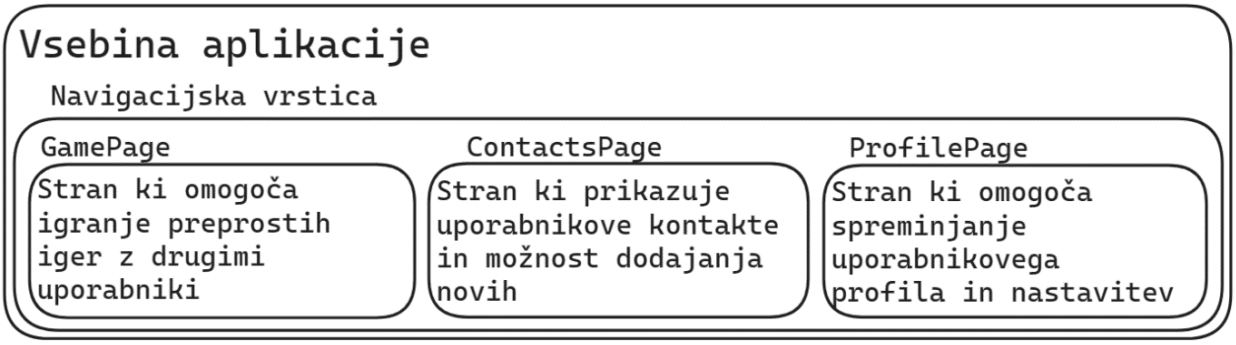


Shema 1: Inicializacija aplikacije

ter datum rojstva in potrdil svojo elektronsko pošto. Potem ga bo usmerilo na del imenovan Start, ki bo poskrbel za pripravo potrebnih podatkov. Ko je aplikacija pripravljena se bo prikazala stran Home, ki skrbi za prikaz navigacijske vrstice in potek podatkov med vsebinskimi stranmi.

3.3.2 Vsebina aplikacije

Vsebino aplikacije navigacijska vrstica loči na tri dele: GamePage(stran z igrami), ContactsPage(stran z kontakti) in ProfilePage(stran z profilom). Med temi tremi stranmi bo mogoče prehajati z horizontalnim potegom levo in desno ali s klikom na določeno ikono na navigacijski vrstici.



Shema 2: Vsebina aplikacije

3.3.2.1 Igralna stran (GamePage)

Stran GamePage bo omogoča igranje preprostih iger z drugimi uporabniki, ki so v kontaktih. Igre bodo imele isto strukturo v podatkovni bazi a bodo imele drugačen način zapisa podatkov.

3.3.2.2 Stran s kontakti (ContactsPage)

Stran ContactsPage bo vsebovala možnost iskanja uporabnikov po uporabniškem imenu, ogled blokiranih oseb in stolpec vseh kontaktov uporabnika. Ob stisu na kontakt se bo odprla stran ChatPage kjer bo potekal pogovor z to osebo. Na strani ChatPage bodo navedene tudi možnosti pošiljanja slik ter datotek, odstranitev in blokiranje kontakta. Ob tiščanju posameznega sporočila se bodo pokazale možnosti:

- **Reply:** možnost odgovoritve na izbrano sporočilo,
- **Copy:** kopiranje vsebine sporočila
- **Edit:** urejanje vsebine sporočila
- **Delete:** brisanje sporočila¸

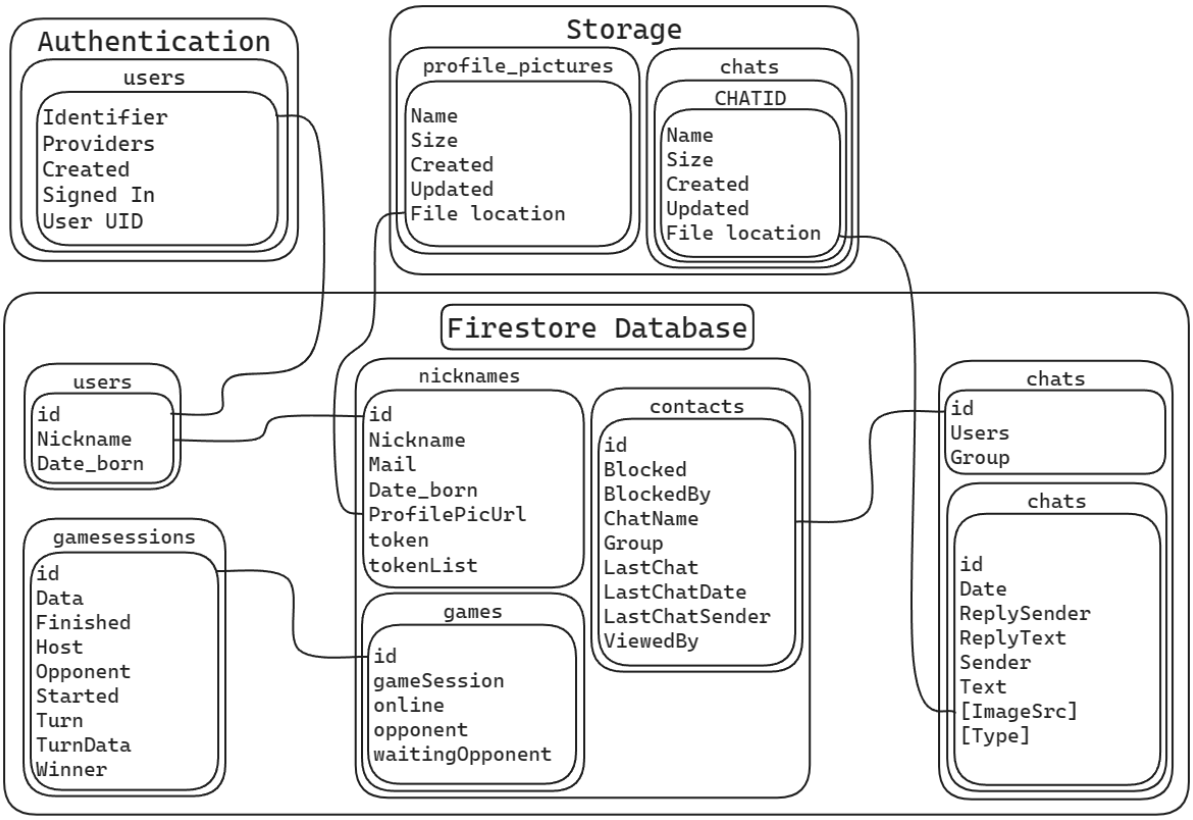
3.3.2.3 Profilna stran (ProfilePage)

Profilna stran bo prikazovala podatke o profilu prijavljenega uporabnika in možnosti za urejanje profila, nastavitve in odjavo iz profila. V urejanju profila

lahko bo možno spreminjanje osnovnih nastavitev profila. Nastavitve bodo omogočale spreminjane teme aplikacije v temno oziroma svetlo temo in spreminjanje velikosti sporočil.

3.3.3 Spletne storitve

Spletne storitve vključujejo Firebase Authentication, Firebase Firestore in Firebase Storage. Storitve sem povezal kar v aplikaciji ker nimam backend-a.



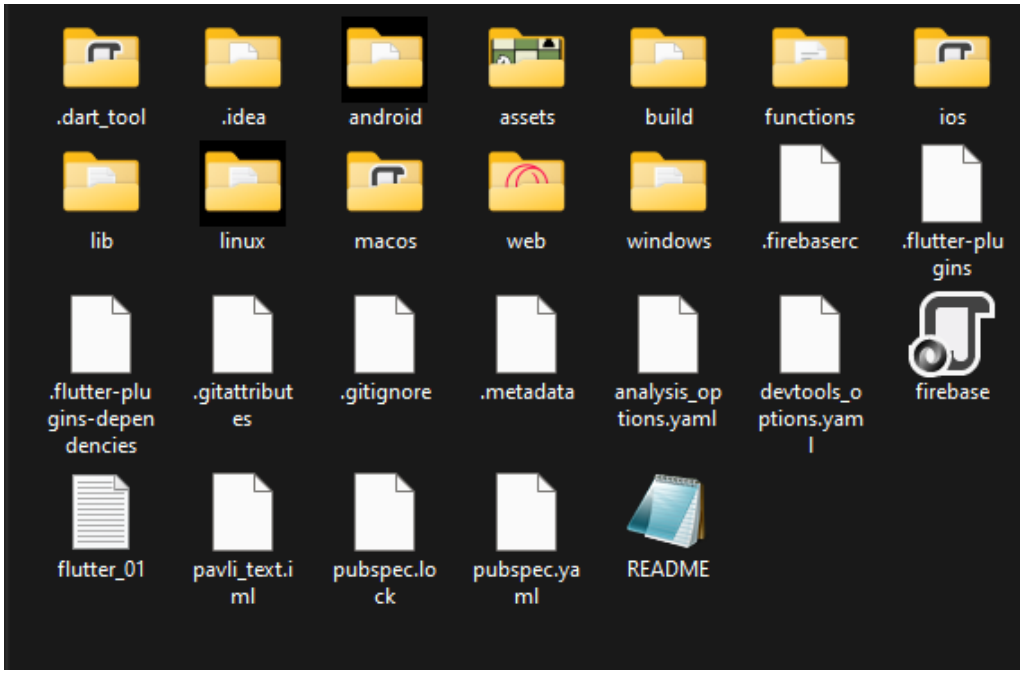
Shema 3: Shema spletnih storitev

3.4 Izvorna koda

Izvorna koda aplikacije se nahaja v mapi pavli_text, ki je deluje kot Flutter projekt. Ime projekta ne sme vsebovati velikih tiskanih črk, zato je ime projekta vse z malo in podčrtaj namesto presledka. Programsko okolje Flutter samo naredi vse potrebne datoteke in podmape potrebne za delovanje.

3.4.1 Mapa projekta

Izvorna mapa projekta »pavli_text« vsebuje veliko podmap in datotek, ki so bili večinsko generirani s strani programskega okolja. Med delom na projektu sem dostopal predvsem do datotek v mapi »lib« kjer se nahajajo datoteke programske kode Dart, ki so glavni vir elementov v aplikaciji. Vse datoteke v »lib« razen »main.dart« sem ustvaril jaz in v njih tudi pisal programsko kodo. Poleg vsebine mape »lib« so pomembne tudi datoteka »pubspec.yaml« in vsebina map »android« kjer so definirane sistemske nastavitve in sredstva za delovanje aplikacije na operacijskem sistemu Android, »assets« kjer so shranjene slike in »build« kamor se shranijo različice zgrajene aplikacije.



Slika 4: Vsebina mape projekta

3.4.2 Knjižnice

Aplikacija PavliText uporablja 25 knjižnic pridobljenih iz uradne spletne strani Flutter-ja. To so:

- **firebase_auth: ^4.2.2**
  - Vsi potrebni razredi potrebni za uporabo storitve Firebase Authentication
- **cloud_firestore: ^4.4.5**
  - Vsebuje potrebne razrede za uporabo podatkovne baze Firebase Firestore
- **firebase_storage: ^11.2.6**
  - Vsi potrebni razredi za uporabo shrambo Firebase Storage
- **firebase_messaging: ^14.3.0**
  - Vsi potrebni razredi za uporabo storitve Firebase Messaging
- **firebase_core: ^2.8.0**
  - Omogoča vzpostavitev povezave z internetnimi storitvami Firebase
- **flutter_local_notifications: ^16.3.3**
  - Knjižnica, ki omogoča prikaz notifikacij
- **http: ^1.2.0**
  - Knjižnica, ki omogoča izvajanje različnih http zahtev
- **date_time: ^0.10.0**
  - Vsebuje potrebno kodo za različne uporabe časa
- **animated_bottom_navigation_bar: ^1.2.0**
  - Razred, ki vsebuje widget navigacijske vrstice z možnostjo animacij
- **flutter_notification_channel: ^2.0.0**
  - Omogoča kreiranje notifikacijskih kanalov(notification channel)
- **plain_notification_token: ^0.0.4**
  - Knjižnica, ki omogoča pridobitev žetona s katerim se lahko pošlje notifikacije
- **path_provider: ^2.0.15**
  - Knjižnica, ki omogoča iskanje lokacij v datotečnem sistemu
- **email_validator: ^2.0.1**
  - Knjižnica za potrjevanje elektronskih naslovov

- **flutter_profile_picture: ^2.0.0**
  - Widget za prikazovanje profilne slike
- **adaptive_theme: ^3.4.1**
  - Knjižnica, ki omogoča preklapljanje med temno in svetlo temo aplikacije
- **awesome_select: ^6.0.0**
  - Knjižnica widget-ov za obrazce
- **image_picker: ^1.0.7**
  - Knjižnica, ki omogoča izbiro slike in pretvorbo slike v programski objekt
- **image_picker_android: ^0.8.9+3**
  - Knjižnica potrebna za pravilno delovanje knjižnice image_picker na operacijskem sistemu image_picker
- **rxdart: ^0.27.7**
  - Knjižnica, ki izboljša zmožnosti programskih tokov(stream)
- **flutter_document_picker: ^5.2.3**
  - Knjižnica, ki omogoča izbiro datoteke in njeno pretvorbo v programski objekt
- **downloadsfolder: ^0.1.1**
  - Knjižnica, ki omogoča zapisovanje datotek v sistemsko mesto za prenose
- **animate_gradient: ^0.0.2+1**
  - Widget za prikazovanje animiranega ozadja

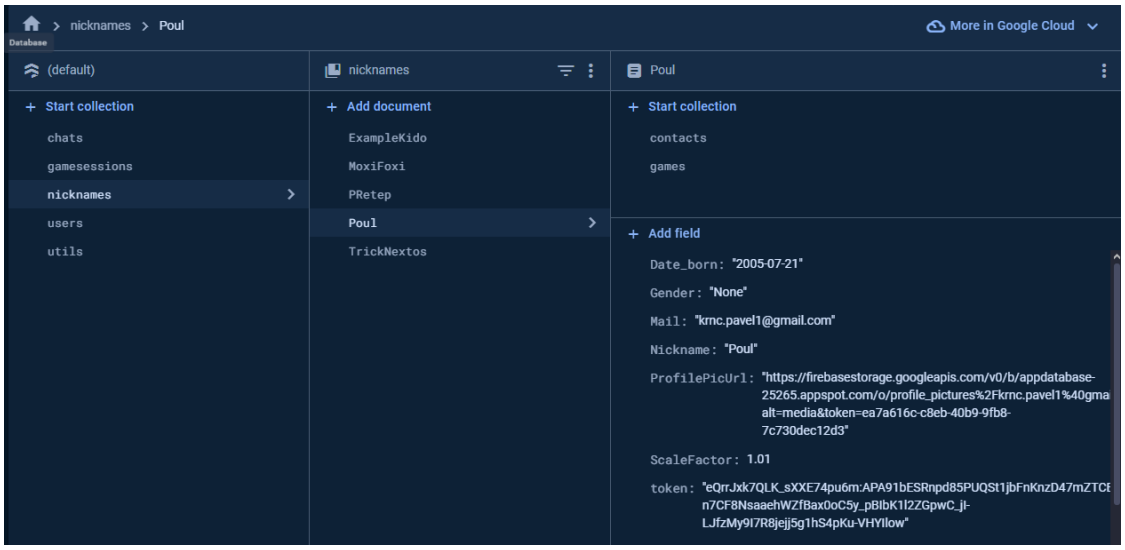
3.4.3 Slike

V aplikaciji so 3 slike, ki niso uporabniško ustvarjene:

- **TICTACTOE.png** je ikona za igro TicTacToe
  - Shranjeno v mapi »assets«
- **CHESS.png** je ikona za igro Chess
  - Shranjeno v mapi »assets«
- **ic_launcher1.png** je ikona aplikacije
  - Shranjeno v mapi pavli_text\android\app\src\main\res\drawable-v21

3.4.4 Podatkovna baza Firestore

Aplikacija za delovanje uporablja podatkovno bazo Firestore storitve Firebase, ki ima svoj »backend« kar omogoča razvijalcem da do podatkov dostopajo direktno iz programa. Nastavitve in povezava do podatkovne baze je zapisana v datoteki »firebase_options.dart«. Podatki v Firestore-u so shranjeni v obliki zbirk in dokumentov na primer: V zbirki »nicknames« so shranjeni vsi dokumenti o uporabnikih aplikacije. Recimo da si izberemo dokument uporabnika »Poul« in v njem so zapisani podatki o njem: ime, datum rojstva, notifikacijski žeton itd. V tem dokumentu pa so tudi še nove zbirke: »contacts«, »games« in v teh zbirkah so lahko ponovno novi dokumenti.



Slika 5: Pogled na podatkovno bazo Firestore

Za pridobivanje podatkov se uporablja knjižnica Firebase Firestore. Primer uporabe podatkovne baze iz aplikacije:

Firebase.instance.collection("nicknames").doc("Poul").collection("contacts").get();

Kar pridobi vse kontakte uporabnika Poul.

3.4.5 Notifikacije

Notifikacije so uporabljene na različnih mestih projekta, to so:

- Ob pošiljanju sporočil uporabniku pošlje še notifikacijo
- Ob povabilu v igro Križci in krožci pošlje notifikacijo
- Ob nalaganju datotek se prikaže notifikacija o stanju

Za prikazovanje notifikacij uporabljam knjižnico »flutter_local_notifications«, za poslušanje notifikacijskih tokov uporabljam storitev Firebase Messaging. Notifikacijo pošljem tako da pridobim notifikacijske žetone uporabnika in potem z http post metodo pošljem primeren body in header na spletno mesto https://fcm.googleapis.com/fcm/send.

```
Map<String, Object> body;

body = {
  "to": tokenL,
  "notification": {"title": widget.data["Nickname"], "body": message}
};

var res = await post(Uri.parse("https://fcm.googleapis.com/fcm/send"),
    body: jsonEncode(body),
    headers: {
      HttpHeaders.contentTypeHeader: "application/json",
      HttpHeaders.authorizationHeader:
          "key=AAAAWCL3XpU:APA91bFxP_DGH1VXWWteQB9ov-KBLF3xzGmklUhlgQCMr
```

Slika 6: Koda za pošiljanje notifikacij

3.4.6 Funkcija main

Ko se program zažene se prvo izvede funkcija »main« v datoteki »main.dart« v kateri se izvede prva konfiguracija in zagon aplikacije. Najprej je poklicana funkcija »ensureInitialized()« iz razreda WidgetFlutterBinding, ki poskrbi, da so gradniki (widgeti) dokončno pripravljeni preden jih prikaže. Nato počaka na vzpostavitev povezave z storitvijo Firebase ter pridobitev informacije o barvni temi aplikacije in šele potem pokliče funkcijo »runApp«, ki prejme kot argument gradnik »MyApp« ta pa kot argument prejme stanje barvne teme. Funkcija »runApp« prikaže grafični vmesnik.

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

  final savedThemeMode = await AdaptiveTheme.getThemeMode();

  runApp(MyApp(
    savedThemeMode: savedThemeMode,
  ));
}
```

Slika 7: Funkcija main

3.4.7 Gradnik MyApp

»MyApp« se nahaja v datoteki »main.dart« in je osnovni gradnik aplikacije, ki kot večina drugih gradnikov uporablja »StatefulWidget«, kar pomeni da se ob spremembi stanja osveži/spremeni. Gradnik »MyApp« definira navigacijski ključ, poskrbi za prejemanje potisnih notifikacij in zažene »MaterialApp«, ki se posreduje aplikacijo v gradnik »StatusPage«.

»MyApp« vsebuje tri funkcije:

- **notificationSetup:** Ta funkcija nastavi dovoljenja in poslušalce za prikaz notifikacij.
- **initState:** Ta funkcija se pokliče ob inicializaciji gradnika in pokliče »notificationSetup«.
- **build:** Ta funkcija zgradi gradnik »AdaptiveTheme«, ki krmari temo aplikacije. V njem je zavit gradnik »MaterialApp«, ki skrbi za krmarjenje aplikacije in pokliče gradnik »StatusPage«.

```dart
class MyApp extends StatefulWidget {
  static final GlobalKey<NavigatorState> navigatorKey =
      GlobalKey<NavigatorState>();
  final AdaptiveThemeMode? savedThemeMode;

  const MyApp({super.key, required this.savedThemeMode});

  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  final _messageStreamController = BehaviorSubject<RemoteMessage>();

  void notificationSetup() async {···

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

  @override
  Widget build(BuildContext context) {
    return AdaptiveTheme(
      //light: ThemeData(),
      light: ThemeData.light(useMaterial3: false),
      dark: ThemeData.dark(useMaterial3: false),
      initial: widget.savedThemeMode ?? AdaptiveThemeMode.light,
      builder: (theme, darkTheme) => MaterialApp(
        scaffoldMessengerKey: messengerKey,
        title: 'PavliText',
        theme: theme,
        darkTheme: darkTheme,
        initialRoute: "/",
        routes: {
          "/": ((context) => const StatusPage(title: "PavliText")),
        },
        /*home: MyMainPage(title: 'App'),*/
      ), // MaterialApp
    ); // AdaptiveTheme
  }
```

Slika 8: Primer kode - MyApp

3.4.8 Gradnik SatusPage

Vse kar dela gradnik »StatusPage« je to, da v funkciji »build« uporabi gradnik »StreamBuilder« in storitev Firebase Authentication za pregledovanje stanja če uporabnik prijavljen ali ne. Če je uporabnik prijavljen ga preusmeri na gradnik »VerifyEmailPage« drugače pa na »AuthPage«. Nahaja se v datoteki »main.dart«.

```dart
class StatusPage extends StatefulWidget {
  const StatusPage({super.key});

  @override
  State<StatusPage> createState() => _StatusPageeState();
}

class _StatusPageeState extends State<StatusPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder<User?>(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return const VerifyEmailPage();
          } else {
            return const AuthPage();
          }
        },
      ), // StreamBuilder
    ); // Scaffold
  }
}
```

Slika 9: Koda za preverjanje stanja avtentikacije

3.4.9 Gradnik AuthPage

»AuthPage« se nahaja v datoteki »auth.dart« v podmapi »auth« in preklaplja med prikazovanjem strani za prijavo in registracijo ter skrbi za animacijo, ki se zgodi ob menjavi.

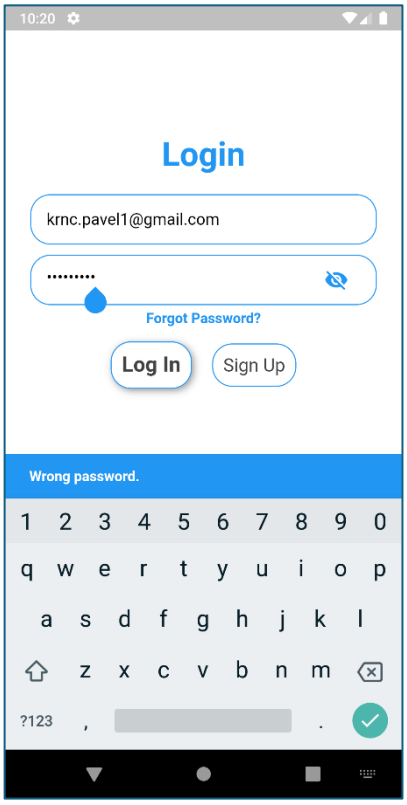
```dart
@override
Widget build(BuildContext context) {
  return AnimatedSwitcher(
    duration: const Duration(milliseconds: 300),
    transitionBuilder: (child, animation) {
      return ScaleTransition(
        scale: animation,
        filterQuality: FilterQuality.medium,
        child: child,
      ); // ScaleTransition
    },
    child: showLogInPage
        ? LogInPage(showRegisterPage: toggleScreens)
        : RegisterPage(showLogInPage: toggleScreens),
  ); // AnimatedSwitcher
}
```

Slika 10: Koda za menjavo med prijavo in registracijo

3.4.10 Gradnik LogInPage

Gradnik »LogInPage« se nahaja v datoteki »login.dart« v podmapi »auth« in omogoča prijavo uporabnikov z uporabo storitve Firebase Authentication. Uporabniški vmesnik zgrajen v funkciji »build« vsebuje dva besedilna polja za elektronski naslov in geslo ter tri gumbe:

- **Forgot Password**, ki uporabnika preusmeri na gradnik »ForgotPasswordPage«
- **Sign Up**, ki preko gradnika »AuthPage« zamenja stran z »RegisterPage«
- **Log In** pa pokliče funkcijo, ki prijavi uporabnika z vnesenim elektronskim naslovom in geslom. Ob napaki prikaže obvestilo z informacijo o napaki



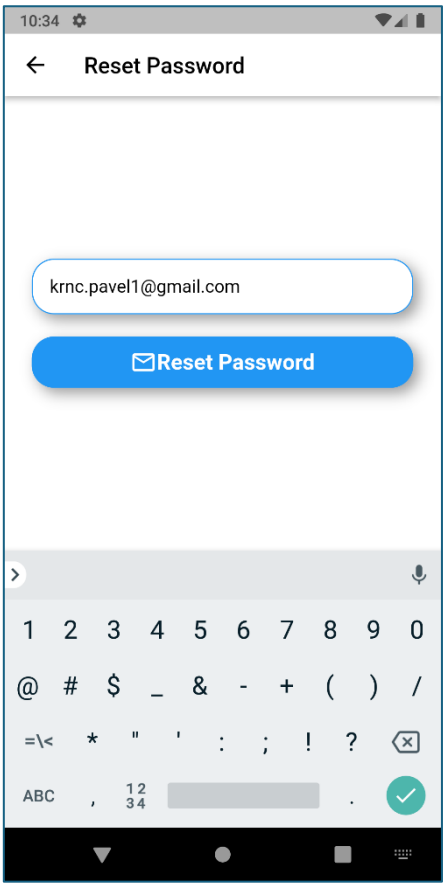
Slika 12: Slika gradnika LogInPage

```
Future login() async {
  try {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim());
  } on FirebaseAuthException catch (e) {
    log(e.code.toString());
    if (e.code == 'user-not-found') {
      Utils.showSnackBar('User not found.');
    } else if (e.code == 'wrong-password') {
      Utils.showSnackBar('Wrong password.');
    } else if (e.code == 'invalid-email') {
      Utils.showSnackBar('Invalid email.');
    } else if (e.code == 'too-many-requests') {
      Utils.showSnackBar('Too many requests.');
    } else {
      Utils.showSnackBar('Something went wrong.');
    }
  }
}
```

Slika 11: Koda za prijavo

3.4.11 Gradnik ForgotPaswordPage

Ta gradnik se nahaja v datoteki »forgot_pasword_page.dart« v podmapi »auth« in omogoča ponastavitev gesla. Vsebuje besedilno polje za elektronski naslov in gumb s katerim storitev Firebase Authentication pošlje elektronsko sporočilo na vneseni naslov z po vezavo do mesta kjer lahko uporabnik spremeni geslo.



Slika 14: Slika gradnika ForgotPasswordPage

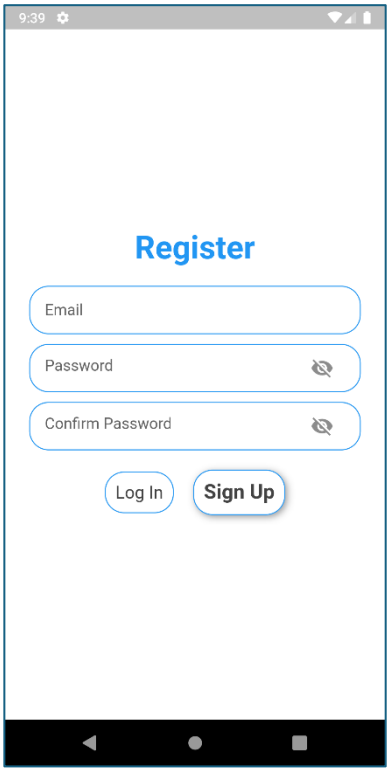
```dart
Future resetPassword() async {
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (context) => const Center(
      child: CircularProgressIndicator(),
    ), // Center
  );
  try {
    await FirebaseAuth.instance
        .sendPasswordResetEmail(email: emailController.text.trim());

    Utils.showSnackBar("Password Reset Email Sent.");
    Navigator.of(context).popUntil((route) => route.isFirst);
    // ignore: unused_catch_clause
  } on FirebaseAuthException catch (e) {
    Utils.showSnackBar("Something went wrong.");
    Navigator.of(context).pop();
  }
}
```

Slika 13: Koda funkcije resetPassword

3.4.12 Gradnik RegisterPage

Gradnik »RegisterPage« se nahaja v datoteki »register.dart« v podmapi »auth« in omogoča registracijo uporabnikov z uporabo storitve Firebase Authentication. Uporabniški vmesnik zgrajen v funkciji »build« vsebuje tri besedilna polja za elektronski naslov, geslo in potrditev gesla ter dva gumba:

- **Log In**, ki preko gradnika »AuthPage« zamenja stran z »LogInPage«.
- **Sign Up** pa pokliče funkcijo, ki registrira uporabnika z vnesenim elektronskim naslovom in geslom. Ob napaki prikaže obvestilo z informacijo o napaki in vsebuje tudi validator, ki pregleduje če so vneseni podatki veljavni.



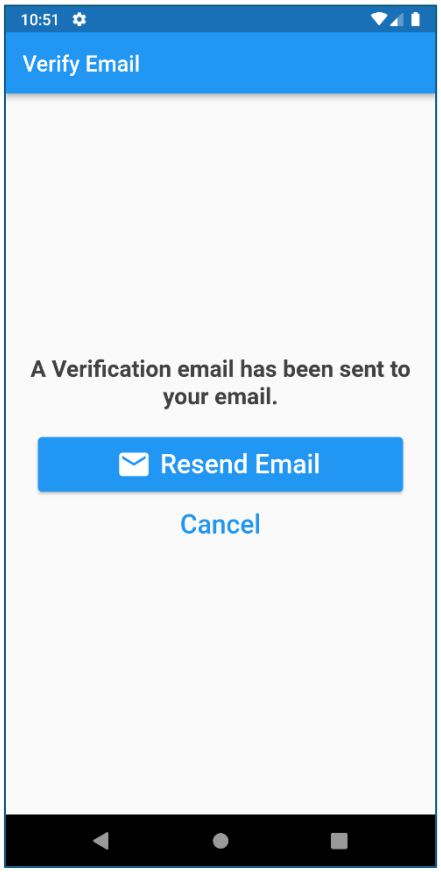
Slika 15: Slika gradnika RegisterPage

```
Future signUp() async {
  final isValid = formKey.currentState!.validate();
  if (isValid) {
    if (_passwordController.text.trim() ==
        _confirmPasswordController.text.trim()) {
      try {
        await FirebaseAuth.instance.createUserWithEmailAndPassword(
            email: _emailController.text.trim(),
            password: _passwordController.text.trim());
      } on FirebaseAuthException catch (e) {
        Utils.showSnackBar(e.message);
      }
    } else {
      Utils.showSnackBar("Passwords do not match.");
    }
  } else {
    Utils.showSnackBar("Something went wrong.");
  }
}
```

Slika 16: Koda funkcije signUp

3.4.13 Gradnik VerifyEmailPage

Ta gradnik se nahaja v datoteki »forgot_password.dart« v podmapi »auth« in je namenjen temu, da preverja ali je elektronski naslov že potrjen. Ob inicializaciji gradnika ko se izvede funkcija »initState« preveri če je elektronski naslov potrjen in če ni pokliče funkcijo »sendVerificationEmail« kar pošlje novo elektronsko sporočilo za potrditev elektronskega naslova ter vzpostavi časovnik, ki vsake tri sekunde pokliče funkcijo »checkEmailVerified« kar ponovno preveri če je elektronski naslov potrjen. Vsebuje gumb »Resend Email«, ki poliče funkcijo »sendVerificationEmail« in gumb za prekinitev, ki odjavi uporabnika iz aplikacije in ga usmeri na gradnik »LogInPage«. Ko je elektronski naslov potrjen preusmeri uporabnika na gradnik »ProfileSetup«.



Slika 18: Slika gradnika VerifyEmailPage

```dart
@override
void initState() {
  super.initState();
  isEmailVerified = FirebaseAuth.instance.currentUser!.emailVerified;

  if (!isEmailVerified) {
    sendVerificationEmail();

    timer = Timer.periodic(
      const Duration(seconds: 3),
      (_) => checkEmailVerified(),
    ); // Timer.periodic
  }
}

Future checkEmailVerified() async {
  await FirebaseAuth.instance.currentUser!.reload();

  setState(() {
    isEmailVerified = FirebaseAuth.instance.currentUser!.emailVerified;
  });

  if (isEmailVerified) timer?.cancel();
}

Future sendVerificationEmail() async {
  try {
    final user = FirebaseAuth.instance.currentUser!;
    await user.sendEmailVerification();
  } catch (e) {
    Utils.showSnackBar(e.toString());
  }
}
```

Slika 17: Koda funkcij v gradniku VerifyEmailPage

3.4.14 Gradnik ProfileSetup

Gradnik »ProfileSetup« se nahaja v datoteki »profile_setup.dart« v podmapi »auth« in je zadolžen za nastavitev podatkov profila če to že ni narejeno. Vsebuje obrazce za vnos uporabniškega imena, datum rojstva, možnost izbire profilne slike in gumb »Next« za potrditev. Ima povezavo do Firebase storitev Authentication, Firestore Database in Firebase Storage. Gradnik ima poleg funkcije »build«, ki skrbi za izris stvari na zaslonu in »initState« še tri funkcije:

- **setUserName:** Ta funkcija se izvede ko uporabnik klikne gumb »Next« in v podatkovni bazi Firestore v kolekciji »users« naredi nov dokument z imenom kakršen je elektronski naslov uporabnika in v njem shrani datum rojstva in uporabniško ime. Potem v kolekciji »nicknames« naredi nov dokument poimenovan po uporabniškem imenu s podatki: uporabniško ime, datum rojstva, elektronski naslov in url do profilne slike. Na koncu še pokliče funkcijo »checkIfExists«.

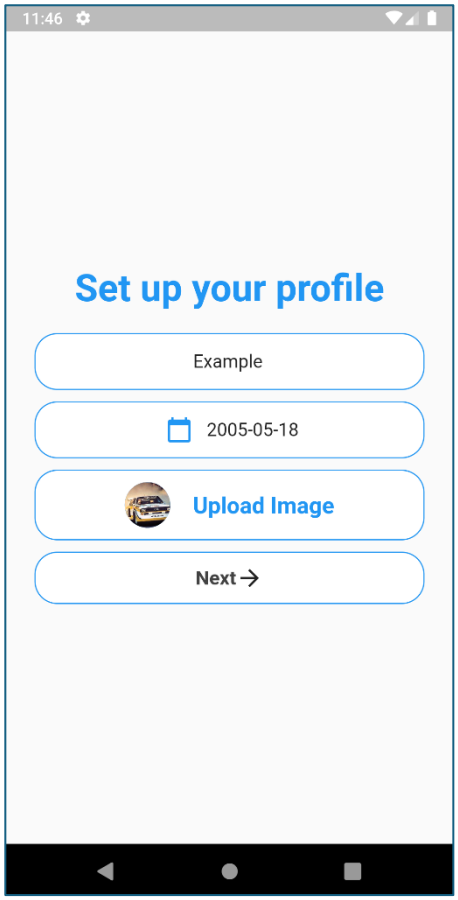
```
void setUserName() async {
  if (_nickNameController.text.trim().length > 15 ||
      _nickNameController.text.trim().length <= 5) {
    Utils.showSnackBar("Your username is either too long or too short.");
    return;
  } else {
    await db
        .collection("users")
        .doc(FirebaseAuth.instance.currentUser!.email!)
        .set({
      "Nickname": _nickNameController.text.trim(),
      "Date_born": _dateController.text.trim(),
    });
    await db
        .collection("nicknames")
        .doc(_nickNameController.text.trim())
        .set({
      "Nickname": _nickNameController.text.trim(),
      "Date_born": _dateController.text.trim(),
      "Mail": FirebaseAuth.instance.currentUser!.email!,
      "ProfilePicUrl": profilePicUrl
    });
    setState(() {
      //refresh
    });
    checkIfExists();
  }
}
```

Slika 19: Koda funkcije setUserName

- **checkIfExists:** Ta funkcija preveri če so podatki že shranjeni v podatkovni bazi in če so postavi logično vrednost »isProfileSet« na true kar bo povzročilo, da

bo uporabnik usmerjen na gradnik »StartPage«. Izvede se ob inicializaciji gradnika in ko kliknemo gumb »Next«.

- **uploadImage:** Ta funkcija je poklicana ko kliknemo na gumb »Upload Image« in z uporabo knjižnice »image_picker« odpre galerijo kjer lahko uporabnik izbere slikoza profil. Ko je slika izbrana jo shrani na storitvi Firebase Storage in v spremenljivko »profilePicUrl shrani povezavo do slike.



Slika 20: Slika gradnika SerProfile

```
void uploadImage() async {
  ImagePicker imagePicker = ImagePicker();
  XFile? file = await imagePicker.pickImage(
      source: ImageSource.gallery, imageQuality: 65);
  if (file == null) {
    return;
  }

  String fileName = user.email!;
  Reference storageRef = FirebaseStorage.instance.ref();
  Reference imageRef = storageRef.child("profile_pictures");
  uploadRef = imageRef.child(fileName);

  printY(fileName);
  await uploadRef.putData(await file.readAsBytes());
  //await uploadRef.putFile(File(file.path));
  String picUrl = await uploadRef.getDownloadURL();
  printY(picUrl);
  setState(() {
    profilePicUrl = picUrl;
  });
}
```

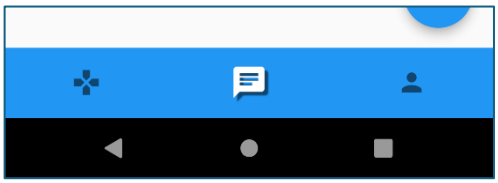
Slika 21: Koda funkcije uploadImage

3.4.15 Gradnik StartPage

»StartPage« se nahaja v datoteki »start.dart«. Ta gradnik pregleda če je aplikacija pravilne različice in pridobi informacije o uporabniku iz podatkovne baze Firestore in žeton naprave, ki se uporablja v storitvi Firebase Messaging. Deluje tako, da se ob inicializaciji gradnika iz funkcije »initState« pokličeta funkciji »setups« in »setup«. »Setups« poskrbi za preverjanje različice aplikacije in pridobitev žetona, »setup« pa za pridobitev informacij o uporabniku.

3.4.16 Gradnik HomePage

Gradnik »HomePage« se nahaja v datoteki »home_page.dart« v podmapi »widget_classes«. Poskrbi za animiran prehod med stranmi vsebine: »ProfilePage«, »ContactsPage« in »GamePage« ter pretok podatkov med njimi. Omogoča premik med vsebinskimi stranmi z bodisi klikom na enega izmed gumbov v navigacijski vrstici ali horizontalnim potegom. Sestavljen je tako da je celoten zaslon dan v gradnik »PageView« v kateremu je list vseh strani in definiran gradnik »BottomNavigationBar«.



Slika 23: Slika gradnika HomePage

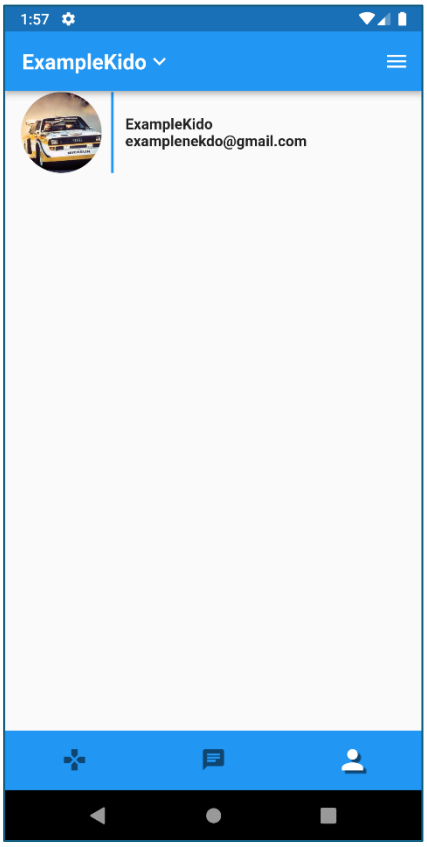
```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
      body: PageView(
        controller: _pageController,
        onPageChanged: _onPageChanged,
        children: <Widget>[
          GamePage(
            data: widget.data,
            setupsList: widget.setupsList,
          ), // GamePage
          ContactsPage(
            data: widget.data,
            setupsList: widget.setupsList,
          ), // ContactsPage
          ProfilePage(
            data: widget.data,
            setupsList: widget.setupsList,
          ) // ProfilePage
        ], // <Widget>[]
      ), // PageView
      bottomNavigationBar: _bottomNavigationBar());
}
```

Slika 22: Koda funkcije buid

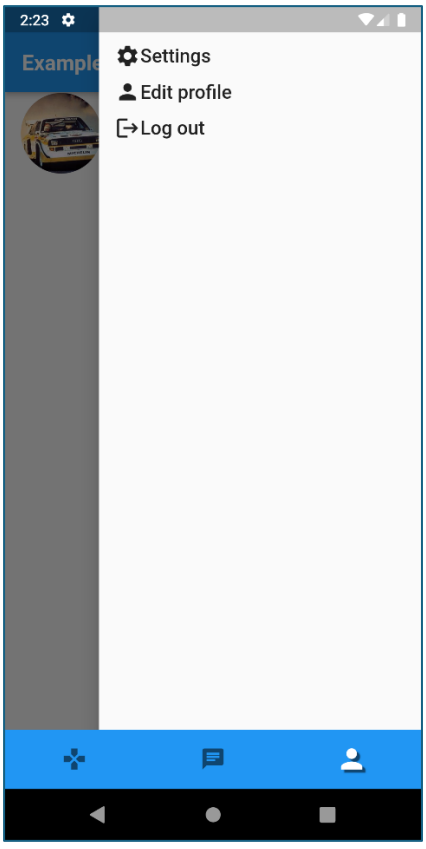
3.4.17 Gradnik ProfilePage

»ProfilePage« se nahaja v datoteki »profile_page.dart« v podmapi »widget_classes« in je eden izmed treh vsebinskih gradnikov te aplikacije. Ob prihodu in izhodi iz te strani se podatki o uporabniku osvežijo. V njem prikaže osnovne podatke o uporabniku in možnosti: urejanje profila, nastavitve in odjava uporabnika. Do teh možnosti lahko dostopamo preko gumba na desni strani naslovne vrstice, ki odpre aplikacijski predal v kateremu so gumbi:

- **Settings:** Preusmeri uporabnika na gradnik »SettingsPage«.
- **Edit profile:** Preusmeri uporabnika na gradnik »EditProfile«.
- **Log out:** Odjavi uporabnika iz aplikacije in ga tako preusmeri na »LogInPage«.



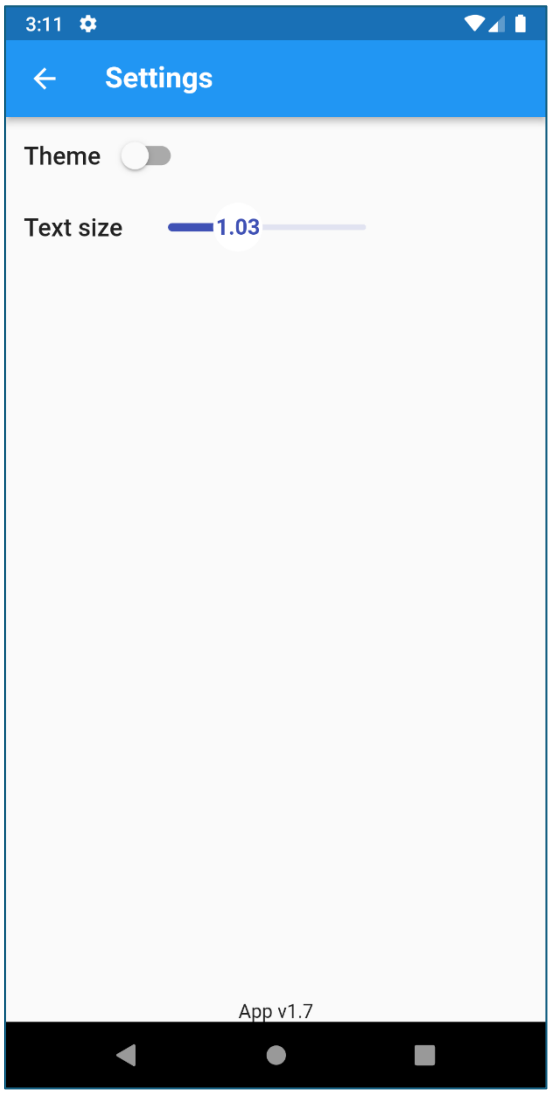
Slika 24: Slika gradnika ProfilePage



Slika 25: Slika predala za več možnosti

3.4.18 Gradnik SettingsPage

»SettingsPage« se nahaja v datoteki »settings_page.dart« in vsebuje možnost za spreminjanje teme aplikacije in drsnik za spreminjanje velikosti pisave pogovorov. Temo aplikacije krmari gradnik »AdaptiveTheme« zato se ta nastavitev spreminja preko njega. Ko zapremo gradnik »SettingsPage« pa se posodobi tudi velikost pisave pogovorov. Za izgradnjo uporablja še gradnik »SliderFb1«, ki naredi drsnik.



Slika 27: Slika gradnika SettingsPage

```
Switch(
  value: AdaptiveTheme.of(context).mode.isDark,
  onChanged: (value) {
    if (value) {
      AdaptiveTheme.of(context).setDark();
    } else {
      AdaptiveTheme.of(context).setLight();
    }
  },
), // Switch
```

Slika 28: Koda za menjavo teme

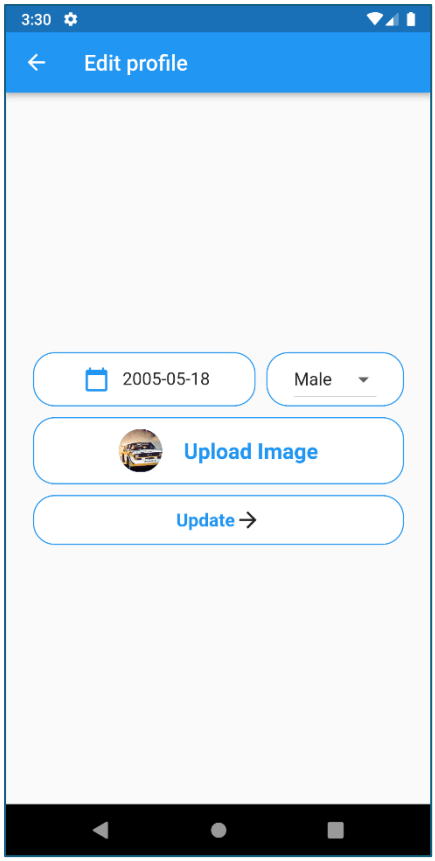
```
@override
void dispose() {
  super.dispose();
  db
      .collection("nicknames")
      .doc(widget.data["Nickname"])
      .update({"ScaleFactor": scaleFactor});
  log(scaleFactor.toString());
}
```

Slika 26: Koda funkcije dispose

3.4.19 Gradnik EditProfile

»EditProfile« se nahaja v datoteki »edit_profile.dart« v podmapi »widget_classes«. Omogoča spreminjanje osnovnih podatkov o profilu. Vsebuje možnost izbire datuma rojstva, izbiro spola in možnost izbire slike. Ob kliku na gumb »Update« se posodobijo podatki na podatkovni bazi. Gradnik ima tri vsebinske funkcije:

- **setups:** Ta funkcija je poklicana iz funkcije »initState«, ki se izvede ob inicializaciji aplikacije in naloži podatke o uporabniku.
- **uploadImage:** Funkcija se izvede ob kliku na gumb »Upload Image« in je naredi isto kot ta funkcija iz gradnika »ProfileSetup«.
- **editProfile:** Ta funkcija vnesene podatke in url slike shrani na podatkovni bazi Firestore.



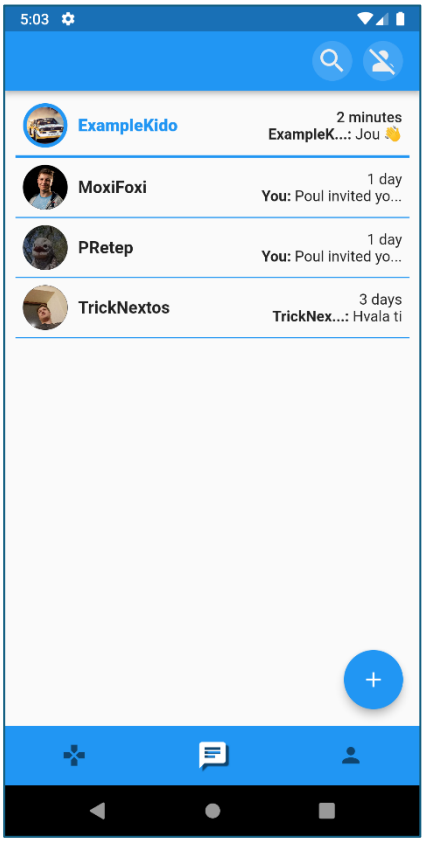
Slika 29: Slika gradnika EditProfile

```
void editProfile() async {
  await db
      .collection("users")
      .doc(FirebaseAuth.instance.currentUser!.email!)
      .update({
    "Date_born": _dateController.text.trim(),
  });
  await db.collection("nicknames").doc(widget.data["Nickname"]).update({
    "Date_born": _dateController.text.trim(),
    "Mail": FirebaseAuth.instance.currentUser!.email!,
    "ProfilePicUrl": profilePicUrl,
    "Gender": dropdownValue
  });
  Navigator.pop(context);
}
```

Slika 30: Koda funkcije editProfile

3.4.20 Gradnik ContactsPage

Gradnik »ContactsPage« se nahaja v datoteki »contacts_page.dart« v podmapi »widget_classes« in je namenjena dodajanju in dostopanju do kontaktov uporabnika. V aplikacijski vrstici sta gumba za iskanje uporabnikov (levo) in gumb za pogled blokiranih uporabnikov (desno). Prvi preusmeri na gradnik »PeopleSearch«, drugi pa na »BlockedContactsPage«. V vsebini je seznam kontaktov, ki so narejeni s gradnikom »ContactWidget« in preusmerijo uporabnika na pogovorno stran »ChatPage«. Spodaj desno je še gumb, ki omogoča dodajanje kontaktov direktno z njihovim uporabniškim imenom.



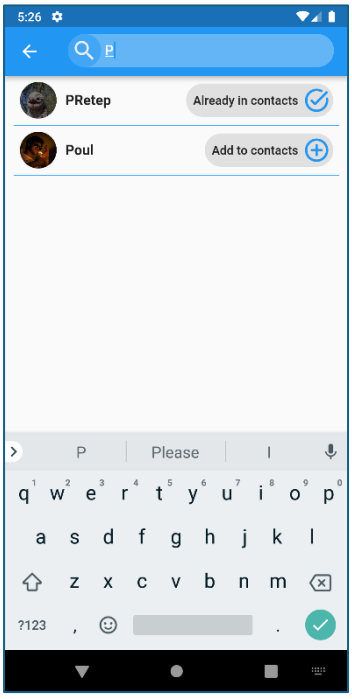
Slika 31: Slika gradnika ChatPage

```
Future<bool> checkIfContactInLst(String personNickname) async {
  bool st1 = true;
  if (personNickname != "") {
    await db
        .collection("nicknames")
        .doc(widget.data["Nickname"])
        .collection("contacts")
        .doc(personNickname)
        .get()
        .then((value) {
      st1 = value.exists;
    });
  }
  return st1;
}
```

Slika 32: Koda funkcije checkIfCintactInLst

3.4.21 Gradnik PeopleSearch

»PeopleSearch« se nahaja v datoteki »widget_helper.dart« in omogoča iskanje uporabnikov po njihovem uporabniškem imenu. Poleg gumba za nazaj še vsebuje besedilno polje v katerega lahko uporabniki napišejo ime oziroma del imena uporabnika, ki ga iščejo. Ko pritisnejo ikono za iskanje se v ozadju izvede iskanje. Iskanje deluje tako, da izpiše tiste uporabnike, ki imajo takšno uporabniško ime, ki se začne z iskalnim nizom ali pa samo drugačne velike in male črke. Ko pridobi podatke o uporabnikih z podobnim imenom so izpisani v vsebino strani kjer jih lahko z pritiskom na gumb »Add to contacts« uporabnik doda.



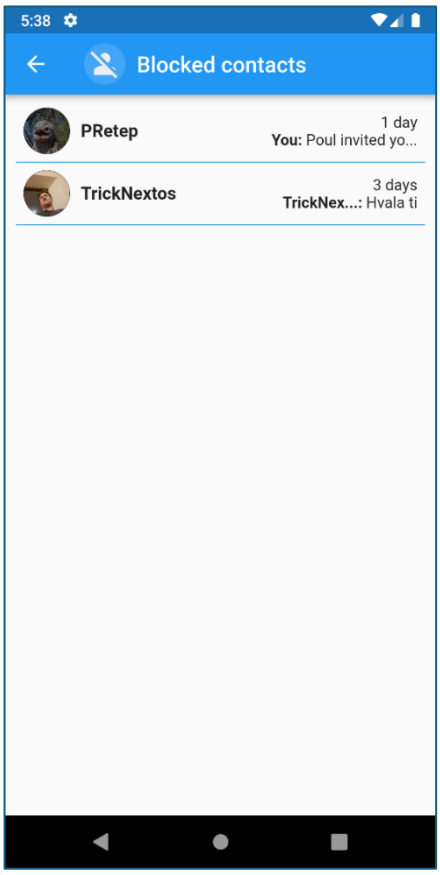
Slika 34: Slika gradnika PeopleSearch

```dart
Future<List<String>> getAllVariations(String input) async {
  List<String> variations = [];

  void generateVariations(String currentString, int index) {
    if (index == input.length) {
      variations.add(currentString);
      return;
    }
    generateVariations(currentString + input[index].toUpperCase(), index + 1);
    generateVariations(currentString + input[index].toLowerCase(), index + 1);
  }
  generateVariations("", 0);
  return variations;
}
```

Slika 33: Koda funkcije getAllVariations

3.4.22 Gradnik BlockedContactsPage

Ta gradnik se prav tako nahaja v datoteki »widget_helper.dart«. V njem se prikažejo kontakti, ki jih je uporabnik blokiral ali pa so oni njega. S klikom na blokiran kontakt se odpre se odpre stran ista kot »ChatPage« samo, namesto pogovora piše kdo je blokiral uporabnika. Uporabnik lahko klikne gumb »Unblock chat« na istem mestu kjer je »Block chat« gumb, da odblokira kontakt ampak samo če je bil on tisti, ki je blokiral drugega.



Slika 35: Slika gradnika BlockedContactPage

```
void unBlockUser() async {
  Navigator.pop(context);
  Navigator.pop(context);
  await db
      .collection('nicknames')
      .doc(widget.data["Nickname"])
      .collection("contacts")
      .doc(widget.contact)
      .update({"Blocked": false, "BlockedBy": ""});
  await db
      .collection('nicknames')
      .doc(widget.contact)
      .collection("contacts")
      .doc(widget.data["Nickname"])
      .update({"Blocked": false, "BlockedBy": ""});
}
```

Slika 36: Koda funkcije unBlockUser

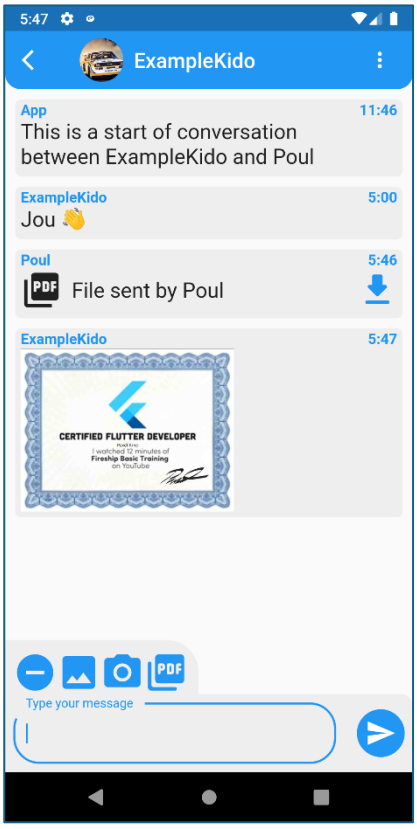
3.4.23 Gradnik ChatPage

»ChatPage« se nahaja v datoteki »chat_page.dart« v podmapi »chats«. Ta gradnik omogoča:

- Pošiljanje besedila
- Pošiljanje slik iz galerije in možnost fotografiranja
- Pošiljanje datotek, ki jih lahko drugi uporabnik prenese
- Odstranjevanje kontakta

- Blokiranje kontakta
- Pošiljanje notifikacij
- Ob tiščanju sporočila omogoča odgovarjanje, kopiranje, urejanje in brisanje določenega sporočila

Deluje tako, da vzpostavi tok podatkov z podatkovne baze s pomočjo gradnika »StreamBuilder«, ki s nato pravilno prikazani v vsebini strani z gradnikom »ChatWidgetStyle«. V aplikacijski vrstici je poleg gumba za nazaj in imena kontakta tudi gumb za več možnosti, ki odpre predal kjer so pokazani možnosti za blokirati ali odstraniti uporabnika iz kontaktov. Spodaj je besedilno polje za napisati sporočilo in puščica za pošiljanje. Zraven so še gumbi za pošiljanje slik in datotek, ki odprejo ali galerijo ali fotoaparat ali pa datoteke. Ob nalaganju ali prenosu datotek se pokaže notifikacija z statusom naloge. Če je poslano več sporočil zapored se združijo skupaj.



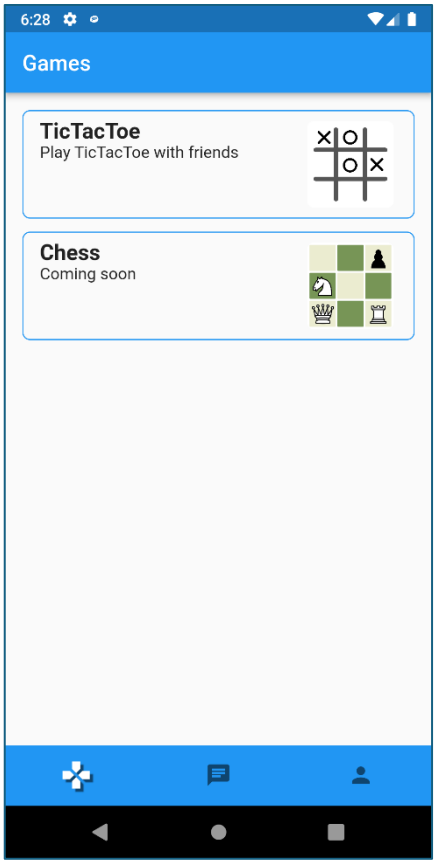
Slika 37: Slika gradnika ChatPage

```
void editChatFunc(DocumentSnapshot<Object?> doc) async {
  Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
  data["Text"] = _editChatController.text;
  if (_editChatController.text.trim() != "") {
    await db
        .collection("chats")
        .doc(chatId)
        .collection("chats")
        .doc(doc.id)
        .update(data);
    unFocusKeyboard();
    Navigator.pop(context);
    setState(() {});
  }
}
```

Slika 38: Koda funkcije editChatFunc

3.4.24 Gradnik GamePage

Gradnik »GamePage« se nahaja v datoteki »game_page.dart« v podmapi »widget_classes« in omogoča igranje iger z drugimi uporabniki. Sestavlja ga samo seznam iger, ki so dodane v aplikacijo. Ko uporabnik klikne na eno igro se mu odpre stran, ki je namenjena tej igri.



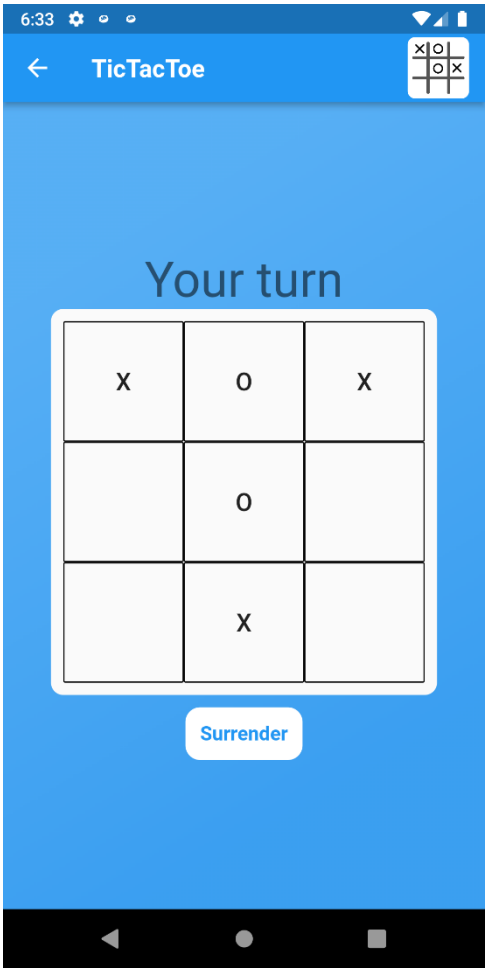
Slika 40: Slika gradnika GamePage

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text("Games")),
    body: SafeArea(
      child: Container(
          padding: const EdgeInsets.all(10),
          child: Column(
            children: [
              GameItem(
                page: Tictactoe(
                  data: widget.data,
                ), // Tictactoe
                title: "TicTacToe",
                img: "assets/TICTACTOE.png",
                data: widget.data,
                text: "Play TicTacToe with friends",
              ), // GameItem
              GameItem(
                page: const Chess(
                    //data: widget.data,
                    ), // Chess
                title: "Chess",
                img: "assets/CHESS.png",
                data: widget.data,
                text: "Coming soon",
              ) // GameItem
            ],
          )), // Column // Container
    ), // SafeArea
  ); // Scaffold
```

Slika 39: Koda funkcije build

3.4.25 Gradnik TicTacToe

»TicTacToe« se nahaja v datoteki »tictactoe.dart« v podmapi »game_widgets«. Omogoča igranje igre križci in krožci z vsemi kontakti. V igro se uporabnike povabi prek gumbov »Start new game« in »Invite another person« kar ti odpre aplikacijski predal kjer so našteti kontakti uporabnika in s klikom na kontakt osebi pošle sporočilo o povabilu in notifikacijo. Ko uporabnik sprejme povabilo osebe se igra začne in ostane dejavna dokler nekdo ne zmaga.



Slika 42: Slika gradnika TicTacToe

```
void gameSetup() async {
  db
      .collection("nicknames")
      .doc(widget.data["Nickname"])
      .collection("games")
      .doc("TicTacToe")
      .set({"online": false});
}

void loadContacts() async {
  db
      .collection("nicknames")
      .doc(widget.data["Nickname"])
      .collection("contacts")
      .get()
      .then((value) {
    var docs = value.docs;
    for (int i = 0; i < docs.length; i++) {
      if (!docs[i].data()["Blocked"]) {
        setState(() {
          contactList.add(docs[i]);
        });
      }
    }
    contactsReady = true;
    setState(() {
      contactsReady = true;
    });
  });
}
```

Slika 41: Koda funkcij gameSetup in loadContacts

4 Zaključek

Moj cilj pri izdelavi aplikacije bil to, da se čim več naučim o programiranju poleg zaključnega izdelka na maturi. Ko sem končal moje delo sem ugotovil, da je bilo pametno s projektom začeti kar eno leto prej saj sem tako lahko delal sproščeno in ne pod pritiskom. Z mojim izdelkom sem zelo zadovoljen saj je veliko več kot sem na začetku načrtoval ampak še bolj sem zadovoljen z znanjem, ki sem ga med delanjem pridobil in upam, da ga bom lahko dobro izkoristil.

5 Viri in literatura

- Flutter (2022). Write your first Flutter app. Pridobljeno s svetovnega spleta Write your first app | Flutter [13.2.2023]
- Wikipedia (2018). Flutter (software). Pridobljeno s svetovnega spleta https://en.wikipedia.org/wiki/Flutter_(software) [13.2.2023]
- Developers (2019). Meet Android studio. Pridobljeno s svetovnega spleta https://developer.android.com/studio/intro [13.2.2023]
- TechTarget (2020). What is Firebase. Pridbljeno s svetovnega sveta https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase [1.5.2024]
- Pub.dev (2019). firebase_auth. Pridobljeno s svetovnega spleta firebase_auth | Flutter package (pub.dev) [10.4.2023]
- Pub.dev (2019). cloud_firestore. Pridobljeno s svetovnega spleta cloud_firestore | Flutter package (pub.dev) [19.5.2023]
- Pub.dev (2019). firebase_storage. Pridobljeno s svetovnega spleta firebase_storage | Flutter package (pub.dev) [21.7.2023]
- Pub.dev (2019). firebase_messaging. Pridobljeno s svetovnega spleta firebase_messaging | Flutter package (pub.dev) [11.9.2023]
- Pub.dev (2019). firebase_core. Pridobljeno s svetovnega spleta firebase_core | Flutter package (pub.dev) [4.4.2023]
- Pub.dev (2019). flutter_local_notifications. Pridobljeno s svetovnega spleta flutter_local_notifications | Flutter package (pub.dev) [11.9.2023]
- Pub.dev (2019). http. Pridobljeno s svetovnega spleta http | Dart package (pub.dev) [10.9.2023]
- Pub.dev (2019). date_time. Pridobljeno s svetovnega spleta date_time | Dart package (pub.dev) [30.5.2023]
- Pub.dev (2019). animated_bottom_navigation_bar. Pridobljeno s svetovnega spleta animated_bottom_navigation_bar | Flutter package (pub.dev) [4.10.2023]
- Pub.dev (2019). flutter_notification_channel. Pridobljeno s svetovnega spleta flutter_notification_channel | Flutter package (pub.dev) [14.3.2024]
- Pub.dev (2019). plain_notification_token. Pridobljeno s svetovnega spleta plain_notification_token package - All Versions (pub.dev) [11.9.2023]

- Pub.dev (2019). path_provider. Pridobljeno s svetovnega spleta path_provider | Flutter package (pub.dev) [16.2.2024]
- Pub.dev (2019). email_validator_auth. Pridobljeno s svetovnega spleta email_validator | Dart package (pub.dev) [11.4.2023]
- Pub.dev (2019). flutter_profile_picture. Pridobljeno s svetovnega spleta flutter_profile_picture example | Flutter package (pub.dev) [15.11.2023]
- Pub.dev (2019). adaptive_theme. Pridobljeno s svetovnega spleta adaptive_theme | Flutter package (pub.dev) [14.9.2023]
- Pub.dev (2019). awesome_select. Pridobljeno s svetovnega spleta awesome_select | Flutter package (pub.dev) [16.11.2023]
- Pub.dev (2019). cloud_functions. Pridobljeno s svetovnega spleta cloud_functions | Flutter package (pub.dev) [13.2.2024]
- Pub.dev (2019). image_picker. Pridobljeno s svetovnega spleta image_picker | Flutter package (pub.dev) [18.9.2023]
- Pub.dev (2019). rxdart. Pridobljeno s svetovnega spleta rxdart | Dart package (pub.dev) [14.2.2024]
- Pub.dev (2019). flutter_document_picker. Pridobljeno s svetovnega spleta flutter_document_picker | Flutter package (pub.dev) [28.2.2024]
- Pub.dev (2019). downloadsfolder. Pridobljeno s svetovnega spleta downloadsfolder changelog | Flutter package (pub.dev) [28.2.2024]
- Pub.dev (2019). animate_gradient. Pridobljeno s svetovnega spleta animate_gradient | Flutter package (pub.dev) [16.3.2024]