Android aplikacija za chat

Hajduković, Zvonimir

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

Sveučilišni studij

Izrada Android aplikacije za chat

Završni rad

Zvonimir Hajduković

Osijek, 2020.

Sadržaj

1. Uvod

Opće poznato je da su ljudi komunikativna bića. Stoljećima se način komunikacije prilagođava situaciji u kojoj se ljudi nalaze i materijalima kojima raspolažu. U početku se na samo nekoliko metara moglo komunicirati s drugom osobom pomoću dvije konzerve i niti, dok danas princip nije toliko drukčiji. Tema ovog završnog rada je izrada Android aplikacije za prijenos tekstualnih poruka među u aplikaciji registriranih korisnika u stvarnom vremenu (engl. *Real-time transmission*). Prijenos poruka se vrši direktnim odašiljanjem inačice poruke i podataka povezanih uz nju prema poslužitelju koji predstavlja skladište registriranih korisnika i njihovih aktivnih poruka i obrnuto. Aplikacija sama vrši transakciju, skladištenje i selekciju podataka koje će biti prikazane korisniku, dok skladište na mreži predstavlja samo skladište bez viših razina funkcionalnosti.

Prilikom izrade aplikacije korišteno je službeno razvojno okruženje Android Studio, unutar kojega je integriran programski jezik Java i jezik za označavanje podataka XML (engl. *eXtensible Markup Language*). Kao rješenje problema skladištenja podataka koji su u ovom slučaju krucijalni za potpun ciklus rada aplikacije iskorišten je poslužiteljski medijski oblak (engl. *Cloud*) Firebase koji razvojnom programeru pruža specifične protokole za komunikaciju između klijenta i poslužitelja.

U nastavku završnog rada slijedi opis po poglavljima: operativni sustav Android, tehnologije korištene u izradi aplikacije unutar razvojnog okruženja Android Studio s prikazom Java i XML kôda, instalacija i postavljanje poslužiteljskog medijskog servera, specifični protokoli za komunikaciju između klijenta i poslužitelja, te prikaz android korisničkog sučelja. Na kraju rada nalazi se zaključak.

1.1. Zadatak završnog rada

Zadatak ovog rada je izraditi samoodrživu android aplikaciju čija će svrha biti korisniku omogućiti komunikaciju s drugim korisnicima iste aplikacije u stvarnom vremenu. Opisati sve potrebne korake izrade aplikacije, postavljanje poslužitelja i distribuciju tekstualnih poruka prema drugim Android

uređajima na lokalnoj ili globalnoj mreži. Također, kroz primjere opisati algoritme koji se koriste prilikom selekcije podataka koji se prikazuju određenom korisniku aplikacije.

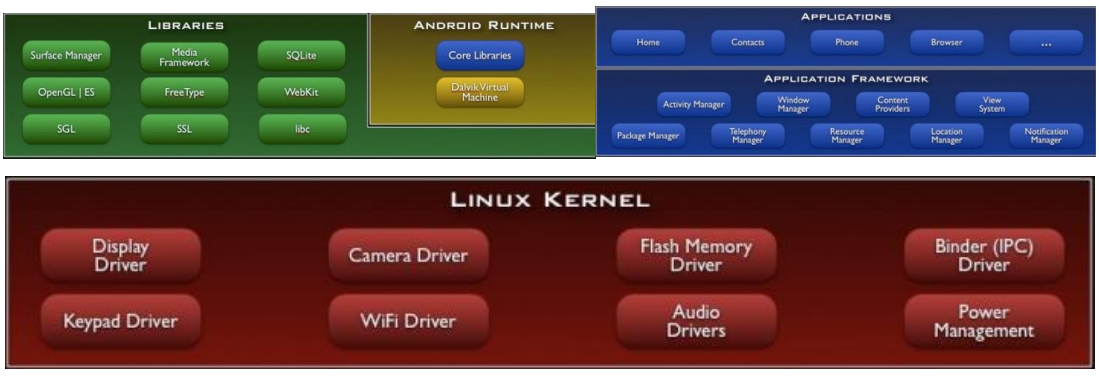
2. OPIS KORIŠTENIIH TEHNOLOGIJA

Tehnologije koje su korištene za razvoj ove aplikacije su opisane u ovom poglavlju. To su operacijski sustav Android, Android Studio, programski jezik Java, opisni jezik XML i medijski poslužitelj Firebase.

2.1. Operacijski sustav Android

Operacijski sustav osnovan je od istoimene kompanije Android Inc. koju su utemeljili Andy Rubin, Rich Miner, Nick Sears i Chris White u listopadu 2003. godine. Isprva, cilj kompanije bio je razvoj operacijskog sustava za digitalne kamere, no ubrzo su uvidjeli da je tržište premalo. Svoj vrhunac dostižu nakon nekoliko godina tajnog rada. Tada je jedino bilo znano da se radi o softveru za mobitele. Google se zainteresirao za rad Android Inc.-a, a još važnije, uvidio je svoju priliku za kupnju karte za utrku na tržištu s pametnim telefonima. U studenome 2007. godine Google osniva OHA-u *(Open Handset Alliance*) s ciljem stvaranja javnog standarda za mobilne uređaje. Android je danas najprodavaniji mobilni operacijski sustav. Zasniva se na Linux 2.6 jezgri i napisan je u programskom jeziku C/C++, ali unatoč tome, većina aplikacija na Androidu napisana je u Javi koristeći pritom Android razvojne programske komponente (engl. *Android Software Development Kit,* SDK). Prema [1] arhitektura sustava Android sastoji se od tri razine (Slika 2.1.):

- Linux jezgra u kojoj se nalaze upravljački programi,
- C/C++ knjižnice iznad jezgre,
- te sloj vidljiv korisniku – Android Runtime



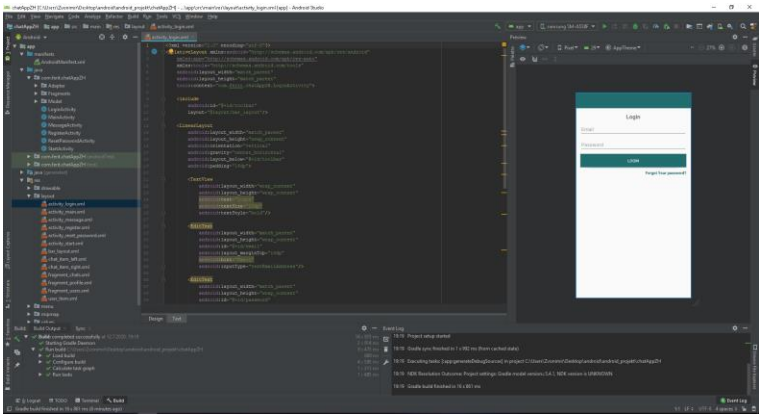
Slika 2.1. Razine Android arhitekture [7]

2.2. Android Studio

Android Studio je službeno integrirano razvojno okruženje za Android platformu. Android Studio je utemeljen na JetBrains IntelliJ IDEA softveru. Na razvojnoj konferenciji Google I/O, objavljen je u svibnju 2013. godine, a prva stabilna verzija, inačica 0.8 izdana je u prosincu 2014. godine.

Andorid Studio podržava MacOS, Windows i Linux operacijske sustave. Prema [2] neke od značajki Android Studia su: Fleksibilna Gradle-bazirana podrška izgradnje, brz i značajkama obogaćen emulator, jedinstveno okruženje gdje se može razvijati aplikacija za sve Android uređaje, prošireni alati za testiranje i distribuciju, integriran pristup vanjskim poslužiteljima poput GitHub-a i Google Cloud platforme, C++ i NDK podrška, ispravljač pogrešaka, alati za praćenje performansi, iskoristivosti resursa, kompatibilnosti verzije sa uređajem te mnoge druge. Ovaj završni rad je gotovo u potpunosti napravljen u Android Studio razvojnom okruženju.

Android Studio razvojno okruženje sadrži **Android SDK** (Software Development Kit) upravitelja (engl. *manager*) [3]. Android SDK (engl. *Software Development Kit*) čini opsežan set razvojnih alata. Alati poput: biblioteke, virtualni uređaj (engl. *emulator*), dokumentaciju, lekcije s primjerima kôda, ispravljač pogrešaka (engl. *debbuger*) i mnogi drugi. Razvojne alate čine komponente koje je moguće skinuti, nadograditi ili čak unazaditi (engl. *downgrade*) radi testiranja starije verzije platforme i kompatibilnosti.



Slika 2.2. Android Studio

2.3. Programski jezik Java

Prema [5], Java je programski jezik objavljen u studenom 1995. godine kojeg su razvili Patrick Naughton, James Gosling i drugi inženjeri u tvrtki Sun Microsystems 1991. godine kao dio projekta Green. Unatoč tome što tvrtka Sun Microsystems posjeduje ime Java kao zaštitni znak, razvojno okruženje moguće je bez plaćanja skinuti s internet poslužitelja spomenute tvrtke. Java jezik je danas jedan od najkorištenijih programskih jezika i procjena te izvješća o broju korisnika kreću se od gotovo 7 do preko 10 milijuna. Sintaksa Jave je slična sintaksi C++-a, pisanje kôda je objektno orijentirano što znači da se piše u klasama, a svaki podataka je objekt, uz iznimku primitivnih tipova podataka kao što su cijeli i decimalni brojevi, *boolean* vrijednosti i znakovi. Unatoč tome što se bazira na C++ sintaksi, Java jezik ne podržava višestruko nasljeđivanje, preopterećivanje operatora (tzv. *polimorfizam*) te globalne ili statičke varijable.

Najveća prednost u odnosu na većinu dotadašnjih programskih jezika je to što su programi izvedivi bez preinaka na svim operativnim sustavima za koje postoji JVM (*Java Virtual Machine [4])*, što razvojnom programeru olakšava posao na način da ne mora prilagođavati aplikaciju operativnom sustavu na kojem se izvodi, a s time i povećava broj korisnika koji u distribuciji mogu koristiti takav softver ili aplikaciju. Također, Java u odnosu na C jezike pruža bolji stupanj sigurnosti i pouzdanosti zahvaljujući JVM-u koji predstavlja zatvoreno okruženje u kojemu se svaki program razvija brže s manje pogrešaka.

Slika 2.3. Java logotip (preuzeto s www.java.com)

2.4. XML

XML (engl. *eXtensible Markup Language*) jezik je za označavanje podataka koji će biti jednostavno čitljivi i ljudima i računalima. Ideja je stvoriti jedan jezik kojemu je princip realizacije jednostavan: oznakama koje ga opisuju i imaju poznato i lako shvatljivo značenje. XML format oznaka je vrlo sličan formatu oznaka u HTML-u (engl. *HyperText Markup Language*), XML je standardiziran jezik. Za njegovu standardizaciju, kao i za standardizaciju drugih sličnih tehnologija poput: CSS-a, HTML-a, XHTML-a, SOAP-a, SVG-a i drugih se brine W3C (*World Wide Web Consortium*). Konkretno, na Android platformi u XML-u opisuje se izgled korisničkog sučelja (veličina, pozicija pojedinih elemenata, boja) te predefinirano ponašanje elemenata ukoliko korisnik vrši interakciju s njima [6].

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end"
    android:text="Forgot Your password?"
    android:layout_marginTop="10dp"
    android:gravity="center"
    android:textStyle="bold"
    android:id="@+id/forgot_password"
    android:textColor="@color/colorPrimaryDark"/>
```

Odsječak koda 2.1. Primjer XML kôda

2.5. Firebase

Firebase je platforma za razvoj web i mobilnih aplikacija. Osnovali su ju James Tamplin i Andrew Lee 2011. godine s ciljem da ostvare održivu bazu podataka za potrebe projekta na kojem su tada radili. Radili su na servisu za komunikaciju sličnom onome koji će biti objašnjen kroz ovaj rad te im je API (engl. *Application program interface*) bio potreban kako bi se brinuo za podatke u stvarnom vremenu. Odlučili su napraviti svoj API. Kroz nekoliko sljedećih godina, mnoge tvrtke uočile su profitabilnost koju nudi Firebase Inc. i donirale novac kako bi sudjelovale u trci za probitak na veliko tržište. Firebase platformu 2014. godine kupio je Google, a Firebase Inc. tim spojen je s dotadašnjim Googleovim timom. Od 2017. godine Firebase platforma nudi velik broj

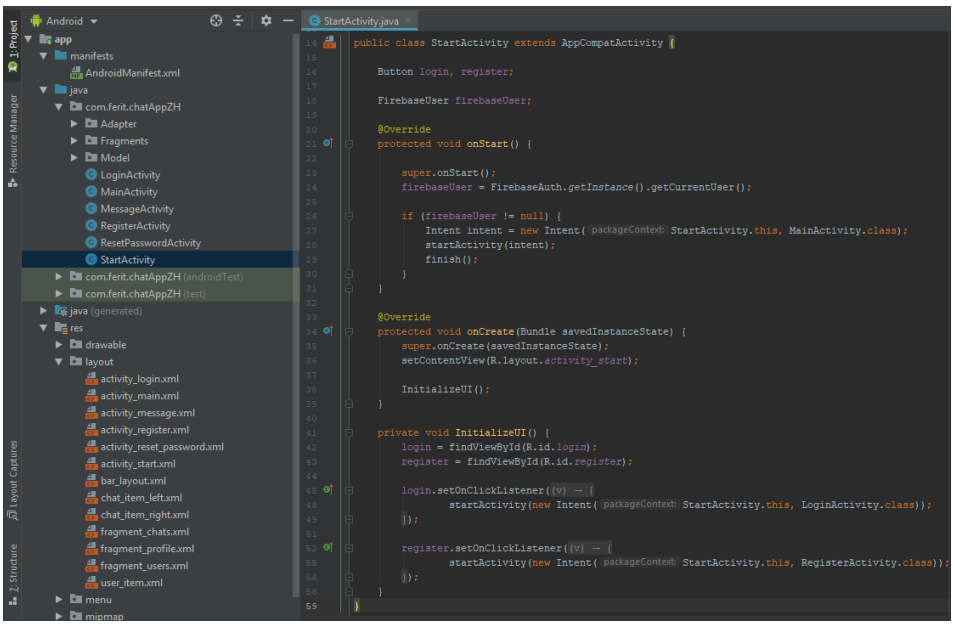
mogućnosti, odnosno servisa, kao što su: Google Analytics, Firebase Cloud Messaging, Cloud Firestore, Firebase Hosting, ML Kit te za ovaj projekt najvažniji servisi: Firebase Authentication i Firebase Realtime Database.

Slika 2.5. Firebase logotip (preuzeto s firebase.google.com)

3. RAZVOJ APLIKACIJE

U ovom poglavlju će detaljno biti opisani svi koraci u kreiranju ove aplikacije uz primjere i objašnjena. Prije samoga kreiranja aplikacije potrebno je skinuti jednu od verzija Android Studia i dobro se upoznati s korisničkim sučeljem i mogućnostima koje pruža. Također, potrebno je dobro poznavanje Java i C/C++ programskih jezika jer su skripte ključan dio kreiranja aplikacije.

Ključni i najkorišteniji koncepti ove Android aplikacije su: knjižnica, Recycler View za omogućavanje prikaza registriranih korisnika te korisničkih poruka u bazi podataka, Adaptera za prilagodbu prenesenih podataka između baze podataka i aplikacije kroz Firebase protokole te mnoge druge.



Slika 3.1. Prikaz svih Java klasa i XML dokumenata od čega je sačinjen ovaj projekt s dijelom koda iz klase *StartActivity*

3.1. Knjižnice

Prije nego li je moguće početi rad na aplikaciji potrebno je u projekt uključiti neke od ključnih implementacija kao što su: RecyclerView, Firebase protokoli za komunikaciju s *Real-time database* serverom te neka od dodatnih knjižnica za ugodniji izgled korisničkoga sučelja aplikacije.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    //implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
    // add the Firebase SDK for Google Analytics
    implementation 'com.google.firebase:firebase-analytics:17.2.0'
    implementation 'com.google.firebase:firebase-auth:19.2.0'
    implementation 'com.google.firebase:firebase-firestore:21.3.1'
    implementation 'com.google.firebase:firebase-database:19.2.0'
    implementation 'com.google.firebase:firebase-storage:19.1.0'
    // additional libraries
    implementation 'com.rengwuxian.materialedittext:library:2.1.4'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'com.google.android.material:material:1.0.0'
    implementation 'androidx.cardview:cardview:1.0.0'
    implementation 'de.hdodenhof:circleimageview:3.0.1'
    implementation 'com.github.bumptech.glide:glide:4.8.0'
    implementation 'com.google.android.material:material:1.0.0'
}

apply plugin: 'com.google.gms.google-services'
```

Odsječak koda 3.1. Implementacija vanjskih knjižnica

3.2. Uvod u rad aplikacije

Prilikom pokretanja aplikacije prva pokrenuta aktivnost je *StartActivity* (slika 3.1.). Takva aktivnost vrši inicijalizaciju dvaju objekata *Intent* klase koja nam služi za prenošenje podataka između različitih aktivnosti u aplikaciji. U većini slučajeva, *Intent* objekti sa sobom prenose

podatke pomoću integriranih metoda *PutExtra()* te u krajnjem slučaju *GetExtra()* kad se takvi podaci koriste u novo nastaloj aktivnosti. U ovome slučaju koristi se najjednostavniji način prelaska iz jedne aktivnosti u drugu. *Intent* objekti koji su inicijalizirani i vidljivi na slici (Slika 3.1.) vode do sljedeće dvije i nešto složenije aktivnosti u nastavku aplikacije, a to su: *RegisterActivity* i *LoginActivity*. Izbor na koju od aktivnosti korisnik želi otići je omogućilo je XML korisničko sučelje koje će biti objašnjeno u selekciji **Struktura Aplikacije** u nastavku.

3.3. Registracija korisnika

Ukoliko korisnik nije od prije registriran korisnik u bazi podataka, kako bi mu bilo omogućeno korištenje aplikacije, potrebno je da se registrira pomoću nekoliko parametara koji će biti pohranjeni u bazu podataka i korisnik će prilikom ponovnog korištenja aplikacije prema njima biti prepoznan. Također, prilikom registracije, postavljeno je nekoliko preduvjeta prije nego li registracija biva prihvaćena i dopusti se daljnje korištenje aplikacije. Preduvjeti su:

1) Sva polja su popunjena
2) Zaporka mora sadržavati minimalno 6 znakova

```
    btn_register.setOnClickListener((v) → {
            String txt_username = username.getText().toString();
            String txt_email = email.getText().toString();
            String txt_password = password.getText().toString();

            if (TextUtils.isEmpty(txt_email) || TextUtils.isEmpty(txt_username) || TextUtils.isEmpty(txt_password)){
                Toast.makeText( context: RegisterActivity.this,  text: "All fields are required", Toast.LENGTH_SHORT).show();
            } else if (txt_password.length() < 6) {
                Toast.makeText( context: RegisterActivity.this,  text: "Password must be at least 6 characters long", Toast.LENGTH_SHORT).show();
            } else {
                register(txt_username, txt_email, txt_password);
            }
    });
}
```

Odsječak koda 3.2. Preduvjeti uspješne registracije korisnika

U ovoj aktivnosti, aplikacija prvi puta dolazi u kontakt s vanjskim izvorom podataka koji se nalaze na Firebase stranici koja nam nudi bazu podataka u stvarnome vremenu. Prilikom prijenosa podataka korišteni su predefinirani protokoli za koje se brine sam Firebase servis koji je uključen

kroz knjižnicu. Pritiskom na tipku „*Register*“ pokreće se skripta koja pomoću već postojećeg protokola i njene metode *CreateUserWithEmailAndPassword(),* preuzima unesene korisničke podatke, provjerava ispravnost podataka te ih kao parametre sigurnim protokolom šalje i pohranjuje u bazu podataka na način vidljiv na slici (Odsječak koda 3.3.). Nakon što je metoda izvršena i dobivena je potvrda uspješne registracije, aplikacija preuzima daljnji posao i pokreće glavni izbornik *StartActivity* koji će biti dodatno objašnjen u nastavku ovoga rada. Ukoliko je potvrda baze podataka neuspješna, korisniku je preko sučelja prikazana kratkotrajna poruka da je registracija s unesenim podacima neuspješna i da pokuša ponovno. Detaljan izgled pohranjenih korisničkih podataka će biti prikazan u nastavku.

```
private void register(final String username, final String email, final String password) {
    auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener((task) → {
                    if (task.isSuccessful()){
                        FirebaseUser firebaseUser = auth.getCurrentUser();
                        assert firebaseUser != null;
                        String userid = firebaseUser.getUid();

                        reference = FirebaseDatabase.getInstance().getReference( path: "Users").child(userid);

                        HashMap<String, String> hashMap = new HashMap<>();
                        hashMap.put("id", userid);
                        hashMap.put("username", username);
                        hashMap.put("imageURL", "default");
                        hashMap.put("password", password);
                        hashMap.put("status", "offline");
                        hashMap.put("search_name", username.toLowerCase());

                        reference.setValue(hashMap).addOnCompleteListener((task) → {
                                if (task.isSuccessful()){
                                    Intent intent = new Intent( packageContext: RegisterActivity.this,MainActivity.class);
                                    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
                                    startActivity(intent);
                                    finish();
                                }
                        });
                    } else {
                        Toast.makeText( context: RegisterActivity.this, text: "You can't register with that email or password", Toast.LENGTH_SHORT).show();
                    }
            });
}
```

Odsječak koda 3.3. Protokol i metoda za registraciju korisnika u bazu podataka

3.4. Prijava korisnika

Ukoliko je korisnik već odradio registraciju kao novi član aplikacije, na početnom izborniku ne odabire „*Register*“ tipku, već „*Login*“. Prilikom inicijalizacije aktivnosti preko *Intent* objekta se

pokreće *LoginActivity* i korisniku je prikazano sučelje slično onome kao i za registraciju, ali s nešto manje potrebnih parametara koji su od samoga korisnika traženi kako bi mu bilo omogućeno daljnje korištenje aplikacije. Kao i prilikom registracije, korisnik unosi parametre za prijavu radi daljnjeg korištenja aplikacije. Specifični Firebase protokoli za ovjeru ispravnosti unesenih podataka provjeravaju sadržava li baza postojećih korisnika traženoga korisnika i šalju aplikaciji potvrdu o uspješnosti pronalaska navedenoga korisnika, ukoliko on postoji. Nakon zaprimljene potvrde o uspješnosti u pronalasku takvoga korisnika, aplikacija preuzima daljnji tok prijave korisnika te pokreće glavni izbornik aplikacije. Ukoliko traženi korisnik ne postoji u bazi registriranih korisnika, korisniku je preko sučelja aplikacije prikazana kratkotrajna poruka o neuspješnosti prijave u aplikaciju te je upućen da pokuša ponovno.

```
        btn_login.setOnClickListener((v) → {
                String txt_email = email.getText().toString();
                String txt_password = password.getText().toString();

                if (TextUtils.isEmpty(txt_email) || TextUtils.isEmpty(txt_password)){
                    Toast.makeText( context: LoginActivity.this,  text: "All fields are required", Toast.LENGTH_SHORT).show();
                } else {
                    auth.signInWithEmailAndPassword(txt_email, txt_password)
                            .addOnCompleteListener((task) → {
                                if (task.isSuccessful()){
                                    Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
                                    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
                                    startActivity(intent);
                                    finish();
                                } else {
                                    Toast.makeText( context: LoginActivity.this,  text: "Authentication failed", Toast.LENGTH_SHORT).show();
                                }
                            });
                }
        });
    }
}
```

Odsječak koda 3.4. Protokol i metoda za prijavu korisnika u bazu podataka

3.5. Glavni izbornik

Uspješnim završetkom opisanih metoda i protokola u prethodna dva odjeljka, za registraciju ili prijavu korisnika, daljnjim tokom aplikacije, korisnika se dovodi do dosad najsloženijeg dijela za razvojnog programera aplikacije, ali vrlo jednostavnog izgleda sučelja s vrlo jasno vidljivim

funkcionalnostima koje su korisniku pružene na korištenje. Većina funkcionalnosti aplikacije obavlja se pozivima iz glavnog izbornika gdje su trenutno omogućene funkcionalnosti poput prikaza svih vidljivih korisnika, pokrenutih razgovora s drugim korisnicima aplikacije, postavki profila ili informacija o trenutno prijavljenom korisniku. Navedene značajke su podijeljene u tri pokretljiva prozora, poznatija još kao i *fragmenti*.

Fragmenti u Java programiranju predstavljaju ponašanje ili odsječak korisničkog sučelja u *FragmentActivityu*. Moguće je u jednu aktivnost iskombinirati više fragmenata kako bi se postiglo više-prozorsko korisničko sučelje te iste fragmente iskoristiti u više aktivnosti. Najjednostavnijim riječima, fragment se može zamisliti kao element koji posjeduje vlastiti životni ciklus, posjeduje i upravlja vlastitim podacima i događajima kojima je moguće upravljati sve dok je aktivnost u kojemu "živi" također "živa".

RecycleView u Java programiranju omogućava fleksibilan prikaz veće količine identičnih inačica koje predstavljaju manje nakupina podataka u prozoru ograničene veličine. Nakupine podataka koje *RecycleView* prikazuje su njegova „djeca" koja posjeduju svoj *indeks* ili poziciju na kojoj se nalaze te je najlakše njima upravljati ako koristimo liste za prikaz takvih podataka.

Glavna zadaća *MainActivitya* je održavati takve fragmente na životu te im omogućiti prostor i povezati njihove funkcionalnosti u jednostavno i jednoznačno korisničko sučelje koje sadrži moderan prikaz fragmenata koje je moguće izmjenjivati uz animaciju kretnje u stranu u kojoj fragment nestaje iz vidljivog dijela aplikacije. Takav oblik rada s fragmentima omogućen je kroz klase *TabLayout, ViewPager* i *ViewPagerAdapter* koje su vidljive na slici u nastavku. Fragmente i njihove funkcionalnosti koje glavni izbornik, odnosno *MainActivity*, održava na životu su:

```
TabLayout tabLayout = findViewById(R.id.tab_layout);
ViewPager viewPager = findViewById(R.id.view_pager);

ViewPagerAdapter viewPagerAdapter = new ViewPagerAdapter(getSupportFragmentManager());
viewPagerAdapter.addFragment(new ChatsFragment(), title: "Chats");
viewPagerAdapter.addFragment(new UsersFragment(), title: "Users");
viewPagerAdapter.addFragment(new ProfileFragment(), title: "Profile");

viewPager.setAdapter(viewPagerAdapter);
tabLayout.setupWithViewPager(viewPager);
```

Odsječak koda 3.5. Deklaracija i upotreba klasa za održavanje ciklusa fragmenata

UsersFragment() koji za zadaću ima brinuti se o prikazu svih korisnika iz baze registriranih korisnika. Metoda i algoritam (Odsječak koda 3.6.) koji se pritom koriste su ta da se na bazu trenutno registriranih korisnika postavlja slušatelj događaja (engl. *Event Listener*) te se prilikom svake promjene na bazi podataka pokreće jedan od algoritama za prikaz korisnika. Glavni algoritam za dohvaćanje korisnika iz vanjske baze podataka i prikaz istih na korisničkom sučelju funkcionira na način da se privremeno inicijalizirana lista, u koju će biti spremljeni pojedinačni korisnički podaci, upotpunjava korisnicima koji zadovolje uvjete algoritma te će se ta ista lista kroz *RecyclerView* prikazati svakome korisniku koji se trenutno nalazi na poziciji ovoga fragmenta. Prilikom dohvaćanja reference baze podataka koristi se metoda *getInstance()* koja je bila omogućena tokom implementacije protokola baze podataka kroz knjižnicu. Kada je referenca baze podataka dohvaćena, algoritam prolazi kroz određene vrijednosti registriranih korisnika te ih dodaje u spomenutu listu korisnika. Postavljen je samo jedan uvjet kojega se algoritam mora strogo pridržavati, a to je da se u listu dodaju svi korisnici osim trenutno prijavljenoga korisnika, što bi značilo da korisnik ne može vidjeti sebe na listi svih trenutno registriranih korisnika jer ne bi imalo smisla da korisnik kontaktira sam sebe.

```
private void readUsers() {

    final FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();
    DatabaseReference reference = FirebaseDatabase.getInstance().getReference( path: "Users");

    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (search_users.getText().toString().equals("")) {
                mUsers.clear();
                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    User user = snapshot.getValue(User.class);

                    assert user != null;
                    if (!user.getId().equals(firebaseUser.getUid())) {
                        mUsers.add(user);
                    }
                }

                userAdapter = new UserAdapter(getContext(), mUsers,  isChat: true,  haveLastMsg: false);
                recyclerView.setAdapter(userAdapter);
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Odsječak koda 3.6. Metoda i algoritam za prikaz korisnika u *UsersFragmentu*()

ChatsFragmetn(). Glavna odgovornost ovog fragmenta prikaz je svih korisnika s kojima je trenutno prijavljeni korisnik imao kontakt, odnosno s kojima je razmijenio barem jednu poruku. Dodatni uvjet u korištenom algoritmu koji također mora biti zadovoljen je taj da je korisniku omogućeno brisanje pojedinačnog prikazanog razgovora bez utjecaja na prethodno objašnjeni fragment. Algoritam u ovome fragmentu funkcionira na gotovo jednak način kao i u prošlome fragmentu. Jedina razlika su uvjeti prema kojima pronalazi korisnike koji zadovoljavaju iste. Prvi algoritam pronalazi sve ljude u bazi podataka s kojima trenutno prijavljen korisnik želi imati prikazan razgovor, a zatim se poziva drugi algoritam koji uspoređuje listu iz prvog algoritma sa svim korisnicima s kojima je trenutno prijavljeni korisnik imao kontakt u vidu međusobnog slanja barem jedne poruke, neovisno o tome je li na poruku odgovoreno ili ne. Svi zadovoljeni uvjeti bivaju spremljeni u privremeno inicijaliziranu listu korisnika i prikazani kroz *RecyclerView* samo u ovome fragmentu.

```
reference = FirebaseDatabase.getInstance().getReference( path: "ChatList").child(fuser.getUid());
reference.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        usersList.clear();
        for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
            ChatList chatList = snapshot.getValue(ChatList.class);
            usersList.add(chatList);
        }

        chatList();
    }

private void chatList() {
    mUsers = new ArrayList<>();
    reference = FirebaseDatabase.getInstance().getReference( path: "Users");
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            mUsers.clear();
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                User user = snapshot.getValue(User.class);
                for (ChatList chatList : usersList) {
                    if (user.getId().equals(chatList.getId())){
                        mUsers.add(user);
```

Odsječak koda 3.7. Metoda i algoritam za prikaz korisnika u *ChatsFragmentu*

ProfileFragment() nešto je složeniji fragment koji trenutno prijavljenom korisniku omogućava učitavanje i uređivanje već učitane slike. Slika se može učitati s Android uređaja na kojemu se koristi aplikacija te biva učitana na *Firebase Storage* servis koji predstavlja vanjsko spremište podataka namijenjeno samo za podatke medijskog sadržaja, u ovome slučaju slike. Uspješno učitana slika prikazana je svim korisnicima aplikacije i stoji pored imena korisnika koji ju je učitao. Algoritam je nešto složeniji iz razloga što se koriste vanjske biblioteke za pomoć pri učitavanju same slike na bazu podataka, a pritom se mora pristupiti integriranom sadržaju ili skladištu medijskog sadržaja na samome uređaju. Prvo se inicijaliziraju klase *Uri* i *StorageTask* pomoću čijih metoda će se upravljati otvorenom toku medija skladišta uređaja i putanji na kojoj se nalazi odabrana slika podržanog formata. Prva provjera u algoritmu je je li slika učitana i je li podržanog formata. U nastavku algoritma se putanja do uspješno odabrane slike šalje sigurnim protokolom do *Storage* servisa na Firebase-u te se slika učitava i pohranjuje na servis. Ukoliko je slika nepodržanoga formata ili je došlo do neke druge nedefinirane greške na uređaju ili u

aplikaciji, prikazuje se poruka o neuspjelom prijenosu i zatvara se tok do medijskog sadržaja uređaja na kojemu je instalirana aplikacija.

```
private void uploadImage() {
    final ProgressDialog pd = new ProgressDialog(getContext());
    pd.setMessage("Uploading");
    pd.show();

    if (imageUri != null) {
        final StorageReference fileReference = storageReference.child(System.currentTimeMillis() + "." + getFileExtension(imageUri));

        uploadTask = fileReference.putFile(imageUri);
        uploadTask.continueWithTask((Continuation) (task) -> {
                if (!task.isSuccessful()) {
                    throw task.getException();
                }
                return fileReference.getDownloadUrl();
        }).addOnCompleteListener((OnCompleteListener) (task) -> {
                if (task.isSuccessful()) {
                    Uri downloadUri = task.getResult();
                    String mUri = downloadUri.toString();

                    reference = FirebaseDatabase.getInstance().getReference( path: "Users").child(fuser.getUid());
                    HashMap<String, Object> map = new HashMap<>();
                    map.put("imageURL", mUri);
                    reference.updateChildren(map);
                    pd.dismiss();
                } else {
                    Toast.makeText(getContext(), text: "Failed!", Toast.LENGTH_SHORT).show();
                    pd.dismiss();
                }
        }).addOnFailureListener((e) -> {
                Toast.makeText(getContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
                pd.dismiss();
        });
    } else {
        Toast.makeText(getContext(), text: "No image selected", Toast.LENGTH_SHORT).show();
    }
}
```

Odsječak koda 3.8. Metode i algoritam za učitavanje profilne slike

3.6. Privatne poruke

Posljednja i vjerojatno najkorištenija aktivnost u aplikaciji je aktivnost za razmjenu poruka s drugim korisnicima aplikacije. Aktivnost za razmjenu poruka ili *MessageActivity* koncipirana je tako da je na određenu granu u Firebase bazi podataka postavljen slušatelj događaja koji protokolima komunicira s aplikacijom samo kada dođe do razmjene poruka. U ovoj aktivnosti razlikujemo dva velika algoritma za čitanje ili prikaz poruka i slanje poruka. Algoritam za slanje poruka je nešto jednostavniji i funkcionira na način da se poziva metoda *sendMessage()*. Prije samoga poziva metode, provjerava se da li je polje predviđeno za upis poruke prazno ili je nešto upisano, a to je uvjet da se spomenuta metoda uopće pozove. Tok algoritma slanja poruke je sljedeći:

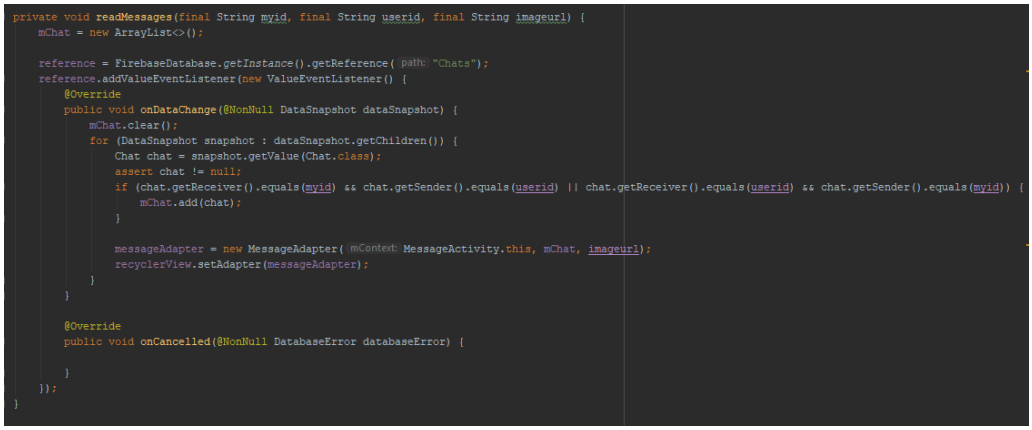
1) Dohvaćanje reference baze podataka
2) Pohranjivanje poruke s određenim elementima:
    a. Pošiljatelj poruke
    b. Primatelj poruke
    c. Tekst poruke
    d. Ostale zadane značajke
3) Dodavanje razgovora u *ChatsFragment()*, pošiljatelju poruke
4) Dodavanje razgovora u *ChatsFragment()*, primatelju poruke

```
private void sendMessage(String sender, String receiver, String message) {
    DatabaseReference reference = FirebaseDatabase.getInstance().getReference();

    HashMap<String, Object> hashMap = new HashMap<>();
    hashMap.put("sender", sender);
    hashMap.put("receiver", receiver);
    hashMap.put("message", message);
    hashMap.put("isSeen", false);

    reference.child("Chats").push().setValue(hashMap);
    final String userid = intent.getStringExtra( name: "userid");

    final DatabaseReference chatRef = FirebaseDatabase.getInstance().getReference( path: "ChatList")
            .child(fuser.getUid())
            .child(userid);
    final DatabaseReference chatRef1 = FirebaseDatabase.getInstance().getReference( path: "ChatList")
            .child(userid)
            .child(fuser.getUid());

    chatRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (!dataSnapshot.exists()) {
                chatRef.child("id").setValue(userid);
            }
        }
```

Odsječak koda 3.9. Metoda i algoritam za slanje poruka

Tok algoritma koji obavlja čitanje i prikaz poruka svaki puta kada se slušatelj događaja na toj grani oglasi i pošalje aplikaciji informacija da je došlo do promjene na bazi podataka:

1) Dohvaćanje reference grane baze podataka na kojoj su spremljene poruke
2) Na bazi podataka se traže poruke gdje se podudaraju jedinstveni korisnički brojevi prijavljenoga korisnika i korisnika s kojim se komunicira
3) Poruke koje zadovoljavaju uvjet se preuzimaju i spremaju u listu
4) Lista se predaje Adapteru koji uređuje takav format poruka
5) Svaka poruka u listi se prikazuje na korisničkom sučelju kao element *RecyclerViewa*



```
private void readMessages(final String myid, final String userid, final String imageurl) {
    mChat = new ArrayList<>();

    reference = FirebaseDatabase.getInstance().getReference( path: "Chats");
    reference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            mChat.clear();
            for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                Chat chat = snapshot.getValue(Chat.class);
                assert chat != null;
                if (chat.getReceiver().equals(myid) && chat.getSender().equals(userid) || chat.getReceiver().equals(userid) && chat.getSender().equals(myid)) {
                    mChat.add(chat);
                }

                messageAdapter = new MessageAdapter( mContext: MessageActivity.this, mChat, imageurl);
                recyclerView.setAdapter(messageAdapter);
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Odsječak koda 3.10. Metoda i algoritam za čitanje i prikaz poruka

3.7. Dodatne značajke aplikacije

Neke od dodatnih značajki aplikacije koje su još vidljive u **Strukturi aplikacije** su:

1) Zaboravljena zaporka
2) Prikaz trenutnog statusa zadnje poslane poruke koji može biti *delievered* ili *seen*
3) Prikaz trenutnog statusa svih korisnika koji može biti *online* ili *offline*
4) Prikaz zadnje poruke s osobama u *ChatsFragment()*
5) Brisanje razgovora razgovora iz prikaza u *ChatsFragment()*
6) Mogućnost pretraživanja po svih osoba u bazi registriranih korisnika po nadimku

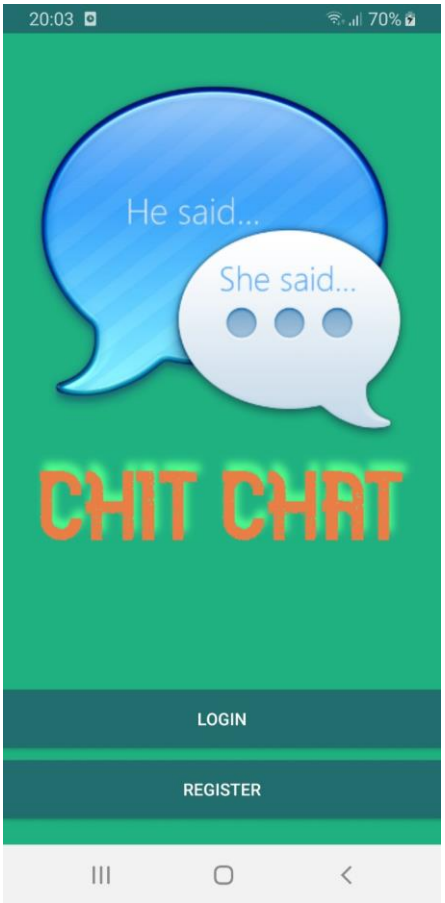
7) Algoritam za automatsku prijavu već postojećeg korisnika (Preskaču se početne aktivnosti i pokreče se glavni izbornik)

4. STRUKTURA APLIKACIJE

U ovom poglavlju pokazat će se izgled aplikacije na uređaju i opisati svi pogledi koji se nalaze u aktivnostima te njihove funkcije.

4.1. Početne aktivnosti

U početnoj aktivnosti aplikacije nalazi se logo aplikacije i dva gumba koji otvaraju preostale dvije početne aktivnosti: aktivnost za registraciju novih korisnika i aktivnost za prijavu već postojećih korisnika. Oba gumba su *button* XML elementi.



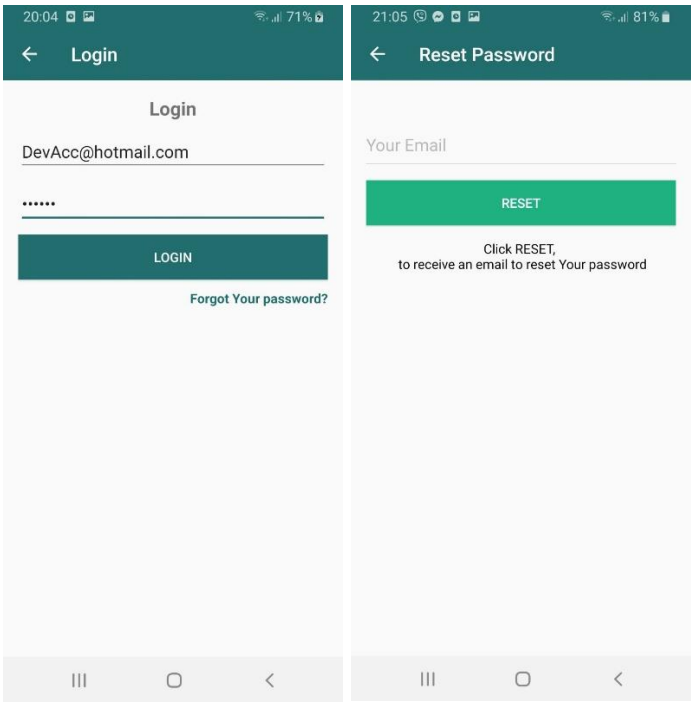
Slika 4.1. Početna aktivnost aplikacije

Ukoliko je na početnoj aktivnosti pritisnut gumb *Register* za registraciju, otvara se aktivnost gdje se od korisnika zahtijevaju potrebne informacije za nastavak u glavni izbornik aplikacije. Elementi koji se zahtijevaju za nastavak su e-mail adresa, proizvoljna zaporka od minimalno 6 znakova i nadimak pod kojim će korisnik biti prikazan ostalim korisnicima aplikacije.

Slika 4.1. Aktivnost registracija

Ukoliko se na početnoj stranici ne izabere gumb za registraciju, nego gumb *Login* za prijavu, otvara se aktivnost u kojoj korisnik unosi svoje podatke za prijavu koje je unosio tokom registracije izuzevši nadimak. Kao dodatna značajka u ovoj aktivnosti se nalazi popularna „zaboravljena zaporka“. Pritiskom na gumb „Forgot Your Password?“, preko *Intent* objekta otvara se aktivnost za povrat

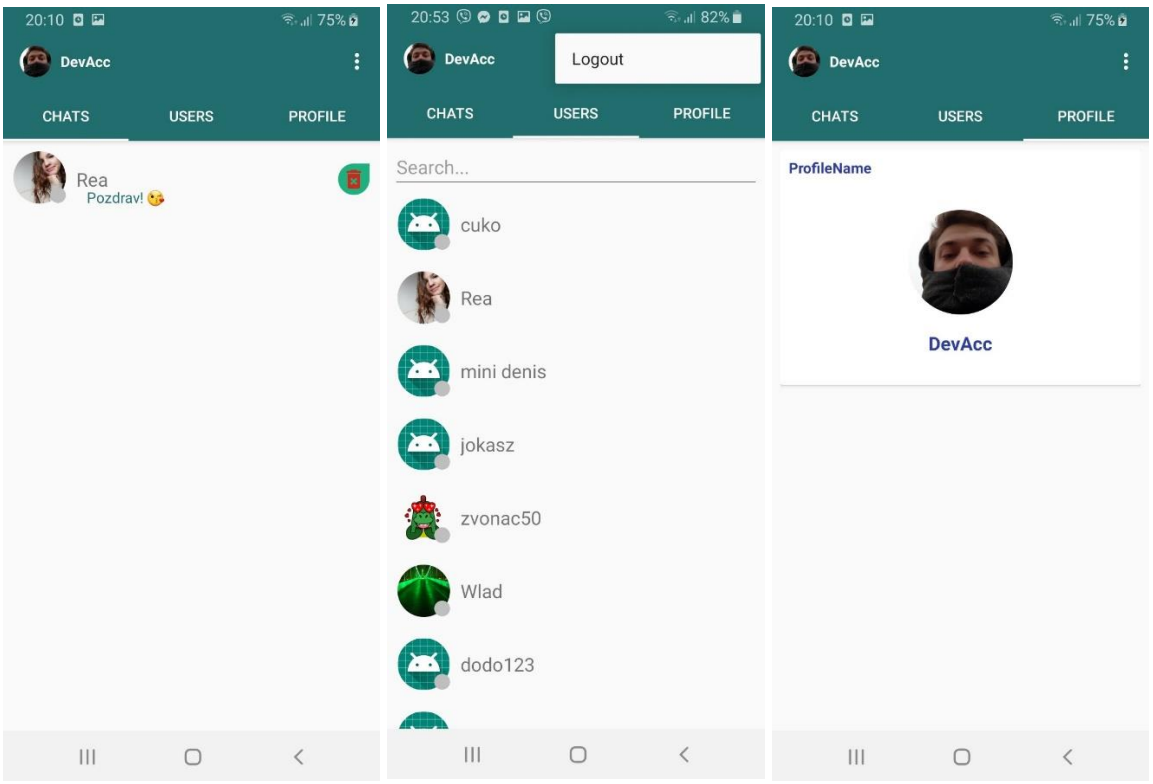
zaboravljene zaporke koja se pritom šalje na e-mail adresu, ukoliko takva postoji u bazi registriranih korisnika.



Slika 4.2. Aktivnost prijava

4.2. Aktivnost glavnog izbornika

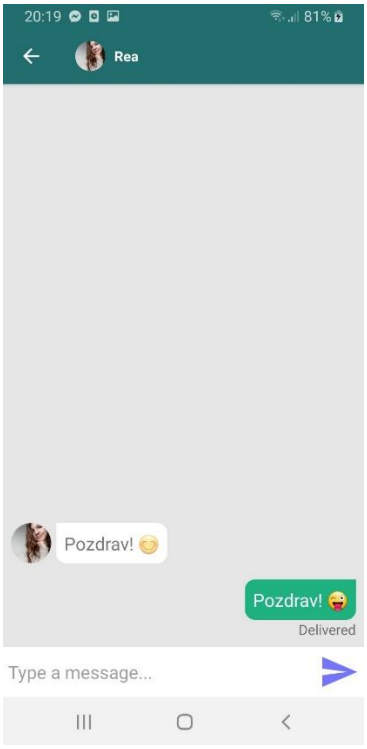
U glavnoj aktivnosti aplikacije nalazi se *TabLayout* na koji je smješten *ViewPager* preko kojega se u konačnici prikazuju fragmenti koji pružaju većinu funkcionalnosti aplikacije. Kretanje po fragmentima može se odvijati na dva načina: kretnjom prsta u stranu ili pritiskom na proizvoljno ime fragmenta u zaglavlju. Prva dva fragmenta upravljaju s elementima jedinstvenih *RecyclerViewova* koji predstavljaju korisnike aplikacije, a zadnji fragment upravlja postavkama profila trenutno prijavljenog korisnika. Razgovore u fragmentu *Chats* je moguće ukloniti pomoću *ImageView* elementa koji pokreće skriptu za uklanjanje elementa *RecyclerViewa* na toj poziciji. Također, iz glavne aktivnosti može se vratiti u početnu aktivnost na način da se odjavi pomoću *Logout* gumba skrivenog iza *menu* XML elementa u gore-desnom kutu.



Slike 4.3., 4.4., 4.5. Glavna aktivnost s najznačajnijim fragmentima

4.3. Aktivnost poruke

U aktivnosti poruka, odnosno *MessageActivityju*, gotovo cijelu aktivnost zauzima *RecyclerView* koji omogućava prikaz poruka koje bivaju dodane u stvarnome vremenu. Ukoliko se u *TextBox* u podnožju unese tekst i pritisne se *ViewImage* desno od polja za unos teksta, pokreću se algoritmi za slanje i čitanje poruka. Iz pogleda pošiljatelja poruke, poruke pošiljatelja su postavljene uz desni rub *RecyclerView* kontejnera, a poruke primatelja lijevo uz njihovu sliku, ukoliko su ju učitali u *Profile* Fragmentu. Dodatna značajka ove aktivnosti je prikaz statusa zadnje poslane poruke koji može biti *delievered* ili *seen*, što bi značilo da je li primatelj poruke vidio zadnju poruku ili ne.



Slika 4.6. Prikaz aktivnost razgovora s drugim korisnikom aplikacije

5. ZAKLJUČAK

Mobilne aplikacije postale su svakodnevnica gotovo u svim fragmentima modernog života. Čovjek je misaono i društveno biće te kao takvo zahtjeva društvo u kojemu može podijeliti svoje misli. Prepreka u tome može nastati u najgorem trenutku, na najgorem mjestu. Ovaj rad zahtjeva podosta istraživanja, predznanja, proučavanja materijala i ideja kako bi se uklonio neizostavan element komunikacije u društvu. Aplikacija je gotovo u potpunosti napravljena u Android Studiu uz pomoć vanjskog spremišta medijskog i podatkovnog sadržaja. Izazovno je razviti ovakvu aplikaciju jer se susreće s dosad nepoznatim konceptima programiranja u Android Studiu i protokolima pruženim kroz vanjske biblioteke. Aplikacija je zamišljena s ciljem da korisnik ostane u komunikaciji s drugim ljudima ukoliko je na neki način spriječen. Ovom aplikacijom korisnik može odlučiti s kime želi komunicirati i da takav komunikacijski kanal održi danas neprocjenjivu privatnost. Trenutno je aplikacija siromašna značajkama koje omogućavaju upravljanje porukama i razgovorima, no kroz daljnji razvoj, može ih se dodati koliko god je potrebno. Tijekom izrade sučelja aplikacije, jako je bitno da korisničko sučelje bude jednostavno, razumljivo i efikasno.

LITERATURA

[1] Wikipedija, Android, dostupno na:

https://hr.wikipedia.org/wiki/Android_(operacijski_sustav),

(srpanj 2020.)

[2] Meet android studio, dostupno na:

https://developer.android.com/studio/intro,

(srpanj 2020.)

[3] Wikipedija, Android SDK, dostupno na:

https://en.wikipedia.org/wiki/Android_software_development

(srpanj 2020.)

[4] Java, dostupno na:

https://go.java.com/,

(srpanj 2020.)

[5] Wikipedija, Java, dostupno na:

https://hr.wikipedia.org/wiki/Java_(programski_jezik)

(srpanj 2020.)

[6] Wikipedija, XML, dostupno na:

https://hr.wikipedia.org/wiki/XML,

(srpanj 2020.)

[7] Android-pit.blogspot.com, Arhitektura Androida, dostupno na:

http://android-pit.blogspot.com/2012/01/arhitektura-androida.html

(srpanj 2020.)

[8] Službena stranica, Firebase, dostupno na:

https://firebase.google.com/

(srpanj 2020.)

SAŽETAK

U ovom završnom radu razvijena je mobilna Android aplikacija koja omogućava razgovor s ostalim korisnicima aplikacije te nazvana je Chit Chat. Korisnici koriste protokole za sigurnu registraciju i prijavu u aplikaciju te su s tim podacima predstavljeni ostatku korisnika aplikacije. Glavni cilj aplikacije je bio omogućiti komunikaciju među registriranim korisnicima te njihove razgovore održati privatnima. Aplikacija je izgrađena u Java programskom jeziku unutar programskog okruženja Android Studio. U radu su objašnjeni ključni algoritmi za održavanje glavnih funkcionalnosti aplikacije te su isti algoritmi kao i izgled samog korisničkog sučelja popraćeni slikama kako bi se prikazalo što je objašnjeno u tekstu.

Ključne riječi: Android, aplikacija, Chat

ABSTRACT

Android chat application

In this bachelor's thesis, an mobile Android application with a name Chit Chat was developed and it allows communicating to all application users. Users are using security protocols for safe registration and login to application database so they can use features of the application. Main goal of developing such application was to enable privated conversations between all registered application clients. Application was made in Java programming language in the Android Studio IDE. In this project, most important algorithms for maintaining main functionalities and the design of user interface were accurately described and accompanied by pictures to show what is explained in the text.

ŽIVOTOPIS

Zvonimir Hajduković rođen je 1.9.1998. godine u Našicama, Republika Hrvatska. Osnovnu školu pohađao je u Osnovnoj školi Vladimira Nazora u Viljevu. Srednju školu je završio u Donjem Miholjcu, smjer opća gimnazija. 2016. godine upisao se na preddiplomski studij Elektrotehničkog fakulteta u Osijeku, smjer Računarstvo.