Osobni dnevnik

Kovačević, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij Računarstva

OSOBNI DNEVNIK

Završni rad

Tomislav Kovačević

Osijek, 2022.

SADRŽAJ

1. UVOD

Ovaj rad opisuje razvoj android aplikacije koja služi kao osobni dnevnik pod nazivom „My Diary“. Kako bi korisnik mogao pristupiti svim funkcionalnostima ove aplikacije nužna je registracija. Nakon uspješne registracije i prijave u sustav, korisnik može stvoriti novu bilješku u kojoj opisuje svoj dan pismenim, zvučnim i slikovnim zapisom. Za određeni datum se može dodati samo jedna bilješka. Na početnom zaslonu prikazuju se sve bilješke dodane od strane korisnika te klikom na pojedinu bilješku može se pregledati i izmijeniti ako je to potrebno. Dugim klikom na bilješku omogućeno je označivanje te brisanje jedne ili više bilješki. Također, ostvarena je mogućnost pretraživanja bilješki po unesenom tekstu. Na zaslonu namijenjenom za statistiku prikazane su informacije o ukupnom broju upisanih bilješki, dodanih slika te zvučnih zapisa. Osim toga, u statističkom dijelu može se vidjeti i grafički prikaz promjene raspoloženja u određenom vremenskom periodu.

Razvojno okruženje (engl. *IDE*) koje je korišteno za razvoj aplikacije je Android Studio. Za pisanje logičkog dijela korišten je Kotlin programski jezik, dok je XML korišten za grafičko sučelje i vizualni prikaz podataka. Za sve *CRUD* funkcionalnosti odnosno, za pohranu, dohvaćanje, uređivanje i brisanje podataka korišten je Firebase Baas (engl. *Backend as a Service*) – konkretno je korišten servis za autentikaciju i NoSQL baza podatka (engl. *Realtime Database*).

Prvo poglavlje je uvod u kojem je ukratko predstavljen zadatak i struktura rada. U drugom poglavlju prikazana su slična rješenja problema kojim se bavi ovaj završni rad. U trećem poglavlju prikazana je teorijska podloga u kojoj se opisuju tehnologije koje su korištene u radu. Četvrto poglavlje predstavlja opis postupka izrade android aplikacije. U petom poglavlju se slikovito predstavlja aplikacija te se u posljednjem poglavlju nalazi zaključak završnog rada.

1.1. Zadatak završnog rada

Napraviti mobilnu aplikaciju koja će imati funkcionalnosti osobnog dnevnika. Aplikacija treba imati mogućnosti zapisa poruka, fotografija, zvučnih zapisa te mogućnost unosa ocjene raspoloženja za svaki dan. Osim toga aplikacija bi trebala imati mogućnost prikaza izvještaja na temelju unesenih podataka za određeni zadani vremenski period.

2. PREGLED SLIČNIH RJEŠENJA

U ovom poglavlju su opisana rješenja odnosno aplikacije koje su trenutačno dostupne na tržištu. Opisano je ukratko koja im je svrha i koje su mogućnosti uspoređujući s aplikacijom koja je produkt ovog završnog rada.

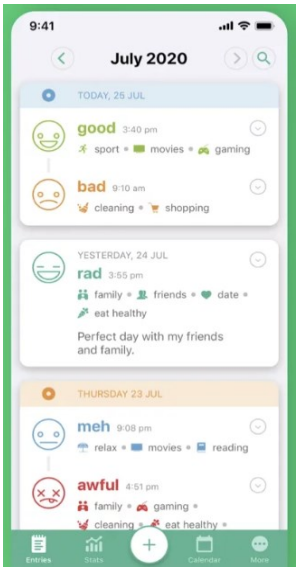
2.1. Diaro

Diaro je platforma za pisanje dnevnika s mnoštvom mogućnosti. Osim mobilnih aplikacija koje su dostupne za Android i IOS operacijske sustave, dostupna je i u web obliku. Aplikacija pruža dodavanje bilješki s mogućnošću odabira kategorije, oznaka, raspoloženja, lokacija te fotografija. Potrebno je napraviti korisnički račun kako bi se podaci mogli uskladiti u svim inačicama aplikacije. Aplikacija je besplatna za korištenje no postoji „PRO“ opcija koja se plaća 6$ godišnje.

Slika 2.1. Prikaz Diaro platforme

2.2. Daylio

Daylio je mobilna aplikacija koja služi za dodavanje bilješki kroz predefinirane kategorije. Neke od kategorija po kojima se stvara bilješka su: raspoloženje, društvenost, hobiji, kvaliteta sna, prehrana, zdravlje itd. Za razliku od prethodno opisane aplikacije i aplikacije koja je izrađena u ovom završnom radu Daylio ne stavlja u fokus tekstualni oblik bilješke nego predefinirane kategorije. Aplikacija je dostupna za Android i IOS operacijski sustav. Na temelju unesenih podataka radi se statistika te na taj način korisnik dobiva povratne informacije.



Slika 2.2. Prikaz zaslona Daylio apliakcije

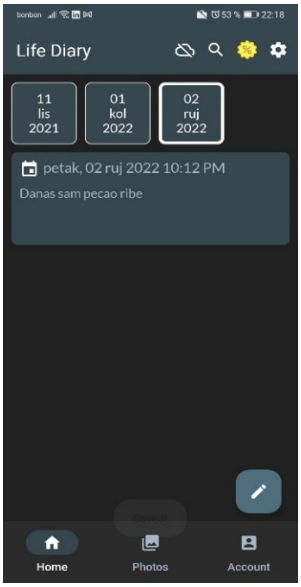
2.3. MyDiary

MyDiary je mobilna aplikacija za pisanje dnevnika. Pri stvaranju bilješke u fokus dolazi tekstualni dio stoga aplikacija pruža bogati uređivač teksta. Uz tekstualni oblik može se priložiti fotografija, naljepnice, smajlići, natuknice, oznake, glasovna poruka i druge mogućnosti. Korisniku je pružena mogućnost uvoza i izvoza bilješki u TXT, PDF i ZIP oblicima.

Slika 2.3. Prikaz logotipa MyDiary aplikacije

2.4. Life Diary

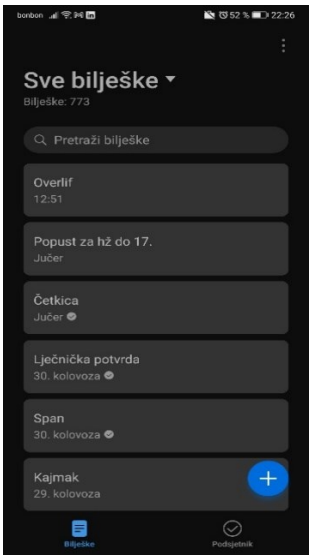
Life Diary je android aplikacija koja služi kao osobni dnevnik. Za razliku od prethodno opisanih, ova aplikacija pruža dosta jednostavnije korisničko sučelje. Uz tekstualni oblik pruža dodavanje fotografije i glasovne poruke. Ova aplikacija je najsličnija aplikaciji koja je produkt ovog završnog rada, razlikuj se u tome što je u Life Diary aplikaciji izostavljen statistički dio.



Slika 2.4. Prikaz zaslona Life Diary aplikacije

2.5. Huawei Notes

Ova aplikacija služi za dodavanje bilješki. S obzirom da nije dnevnik, datum ne dolazi u izražaj kao u prethodnih aplikacijama. Pri dodavanju bilješke mogu se priložiti fotografije, crteži, popisi za provjeru te postoji mogućnost korigiranja fonta.



Slika 2.5. Prikaz zaslona Huawei Notes aplikacije

3. TEORIJSKA PODLOGA

U ovom poglavlju bit će ukratko opisane najvažnije tehnologije i alati koji su korišteni u razvoju android aplikacije.

3.1. Android Studio

Android Studio je integrirano razvojno okruženje, odnosno IDE (engl. *Integrated Development Environment*) bazirano na IntelliJ IDEA. Uz rješenja koja donosi IntelliJ poput moćnog alata za pisanje koda i razvojnih alata, Android Studio nudi dodatne mogućnosti koje povećavaju produktivnost pri razvoju Android aplikacija. Neke dodatne mogućnosti koje nudi Android Studio su: pokretanje sustava bazirano na *Gradle* načinu, brzi emulator s mnoštvom mogućnosti, okruženje za razvoj aplikacija prikladnih svim Android uređajima, alati za testiranje koda, predlošci koda i integracija sa GitHub-om itd. Svaki projekt sastoji se od jednog ili više modula koji sadrže izvorni kod i druge resurse. Sve datoteke zadužene za pokretanje programa su smještene pod *Gradle Scripts* grupacijom, dok svaki aplikacijski modul sadrži sljedeće direktorije: **manifesti** (engl. *manifests*) – sadrže AndoridManifest.xml datoteku, **java** – sadrži Java datoteke sa izvornim kodom, uključujući JUnit test kod, **res** – sadrži sve resurse bez koda, poput XML slojeva, UI stringova i bitmap slika.[1]

IntelliJ IDEA je *IDE* napisan u Java programskom jeziku za razvoj programske podrške pisane u Java, Kotlin, Groovy i drugim JAR-baziranim jezicima. Razvijen je od strane tvrtke JetBrains, njihovi proizvodi namijenjeni su za razvoj komercijalnih proizvoda.[2]

Slika 3.1. Android Studio i IntelliJ IDEA logotip

3.2. Kotlin programski jezik

Kotlin je više-platformski, statički pisan, opće namjenski programski jezik. Kotlin je dizajniran da međusobno surađuje sa Javom, a JVM verzija Kotlina je standardna biblioteka ovisna o *Java Class* biblioteci, dok sučelje za pisanje (engl. *typed interface*) omogućuje da sintaksa bude što više sažeta. Najviše je korišten pisanju aplikacije temeljenim na JVM-u (*Java Virtual Machine*), ali također može prevoditi u JavaScript (npr. za *frontend* web aplikacije bazirane na React-u) ili u izvorni kod (engl. *native code*). Cijenu razvoja Kotlina financirao je JetBrains dok Zaklada Kotlin štiti zaštitni znak „Kotlin“. Google je 2019. godine proglasio Kotlin kao poželjan jezik za razvoj Android aplikacija.[3]

3.3. XML

Proširivi jezik za označavanje (XML) je jednostavni tekstualno-bazirani format za prikaz strukturiranih informacija poput: dokumenata, podataka, konfiguracija, knjiga, zvukova i mnogih drugih. Odijeljen je od starijeg standardnog formata zvanog SGML (ISO 8879), s namjerom da bude prilagođen za korištenje u Web-u. Naizgled je dosta sličan HTML-u. Sintaksa mu je dosta striktna stoga XML alati neće obraditi datoteke koje sadrže greške. XML u današnje vrijeme je korišten u širokom spektru različitih područja. Neke od primjena XML-a: mnoštvo standarda je bazirano na njemu poput *UBL*-a, *UPnP*-a koji se koristi za kućnu elektroniku, za obradu riječi poput *ODF*-a, kod grafičkih formata poput *SVG*-a, ima direktnu podržanost od računalnih programskih jezika i baza podataka.[4]

```
<item>
    <shape>
        <corners android:radius="9dp"></corners>
        <stroke android:width="1dp" android:color="@color/black"></stroke>
        <solid android:color="#FFFFFF"></solid>
    </shape>
</item>
```

Slika 3.2. Primjer XML elemenata

3.4. Firebase

Firebase je BaaS (engl. *Backend as a Service*) platforma za razvoj aplikacija koja pruža *backend* usluge poput baze podataka, pohrane u oblaku, autentikacije, izvještaja o radu, strojnog učenja, konfiguracije na daljinu i mnoge druge.[5]

Dvije usluge koje pruža Firebase su važne za ovaj rad, a to su: *Firebase Realtime Database* i *Authentication*, stoga će oni biti opisani u daljnjem tekstu.

3.4.1. Firebase Realtime Database

Firebase Realtime Database je baza podataka u oblaku (engl. cloud-based).[6] Usluga služi za spremanje i usklađivanje podataka sa NoSQL bazom podataka. Podaci su usklađeni sa svim klijentima u stvarnom vremenu i ostaju dostupni kad aplikacija izgubi povezanost s internetom. Podaci se spremaju u JSON obliku. Baza je neovisna o platformi koja ju koristi, to mogu biti: Android, JavaScript SDK ili Apple platforme i sve će biti istovremeno obavještene o novim promjenama. Za razliku od običnih HTTP zahtjeva, Realtime Database koristi sinkronizaciju podataka – svaki put kad se podaci promjene, svaki povezani uređaj biva obavješten. Također, važno je naglasiti da postoji *Firebase Realtime Database Security Rules*, odnosno, sigurnosna pravila koja osiguravaju da baza podataka ostane sigurna i da se ograniči njen pristup samo na one koji uistinu to trebaju. S obzirom da je NoSQL baza podataka, ima drugačije optimizacije i funkcionalnosti uspoređujući s relacijskim bazama podataka. Realtime Database API (engl. *Application programming interface*) je dizajniran tako da budu dozvoljene samo operacije koje se brzo izvršavaju.

3.4.2. Firebase Authentication

Ova usluga pruža SDK koji je lagan za korištenje te gotove biblioteke korisničkog sučelja za autentikaciju korisnika u aplikaciju. Podržava autentikaciju koristeći lozinku, broj mobilnog telefona, te popularne federalne identitete poput Google-a, Facebook-a, Twitter-a i drugih.

Firebase Authentication je u čvrstoj vezi s ostalim uslugama koje nudi Firebase, a također može biti integriran s *backendom* koji nije u vlasništvu Firebase-a.

3.5. Dodatne biblioteke

Ovaj rad opisuje programsku podršku koja ima razne funkcionalnosti koje ne bi bile moguće ili bi bile puno teže realizirane bez uključivanja dodatnih biblioteka. Osim ugrađenih biblioteka u daljnjem tekstu će biti opisane tri važne biblioteke koje su korištene pri izradi programske podrške.

3.5.1. Hilt

Hilt je biblioteka za Android, čija je svrha pojednostavljenje provedbe ubrizgavanja ovisnosti (engl. *dependency injection*) u projekt. Manualna provedba ubrizgavanja ovisnosti zahtjeva ručno konstruiranje svake klase i njihovih ovisnosti te korištenje spremnika kako bi se omogućilo ponovno korištenje, taj cijeli posao se delegira na Hilt biblioteku. [7]

Hilt pruža spremnike za svaku Android klasu u projektu te upravlja njihovim životnim vjekovima automatski. Hilt je izgrađen na vrhu popularne DI biblioteke pod nazivom Dagger kako bi imao koristi od ispravnosti vremena prevođenja, performansi, skalabilnosti i podržanost od strane Android Studio IDE-a.

3.5.2. Material DateTime Picker

Biblioteka puža dodatne mogućnosti odabira datuma u usporedbi s običnim *DatePicker*-om kojeg pruža Android Studio. Neke od dodatnih mogućnosti su: postavljanje drugačijih tema, postavljanje omogućenih/onemogućenih datuma, postavljanje označenih datuma, uključivanje vibracija i mnoge druge. Autor ove biblioteke po podacima s GitHub-a je Wouter Dullaert.

Ono što je konkretno korišteno u ovom projektu je postavljanje onemogućenih datuma, kako se ne bi mogao odabrati već odabrani datum.

3.5.3. MPAndroidChart

Moćna i jednostavna biblioteka za crtanje grafova dizajnirana za Android. Sadrži sve gotove XML dokumente i klase s metodama spremim za korištenje. U ovom radu ova biblioteka je prvenstveno korištena za prikaz promjene raspoloženja u vremenu. Na apscisi nalazi se vrijeme, a na ordinati se nalazi broj od 1-10 što predstavlja raspoloženje. Autor ove biblioteke po podacima s Githuba je Philip Jahoda.

4. POSTUPAK IZRADE ANDROID APLIKACIJE OSOBNI DNEVNIK

U ovom poglavlju su opisani glavni dijelovi aplikacije popraćeni slikovnim prikazom dijelova koda. Prije svega opisane su krucijalne aktivnosti su nužne za rad aplikacije, a daljnji redoslijed opisivanja je po uzoru na redoslijed kroz kojeg prolazi korisnik kad se prvi susreće sa aplikacijom. Opisan je proces autentikacije u kojeg spada registracija, prijava te promjena lozinke. Nakon toga opisuje se prikaz i unos bilješki te na temelju unesenih podataka napravljen je statistički dio. Opisan je također i zaslon sa osobnim podacima te ostale klase koje služe kako bi se sva logika izvršila bez poteškoća.

4.1. Glavne aktivnosti

U ovom poglavlju opisane su aktivnosti koje su ključne za uspješan rad aplikacije.

4.1.1. Dispatcher aktivnost

Ovo je aktivnost bez XML datoteke (nema grafičko sučelje) koja se pokreće uvijek kad se aplikacija pokrene. U AndroidManifest.xml datoteci na slici 4.1. vidi se da je ova aktivnost definirana kao glavna. U *onCreate* metodi se radi provjera je li korisnik prijavljen u sustav, ako je, odlazi na Home aktivnost, u suprotnom odlazi na aktivnost za prijavu u sustav. S ovakvim ovakvom strukturom projekta se postiže da korisnik nakon što ponovno pokrene mobilni uređaj i dalje ostane prijavljen iz razloga jer su mu podaci ostali spremljeni u dijeljenim preferencijama.

```
<activity
    android:name=".Activities.DispatcherActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

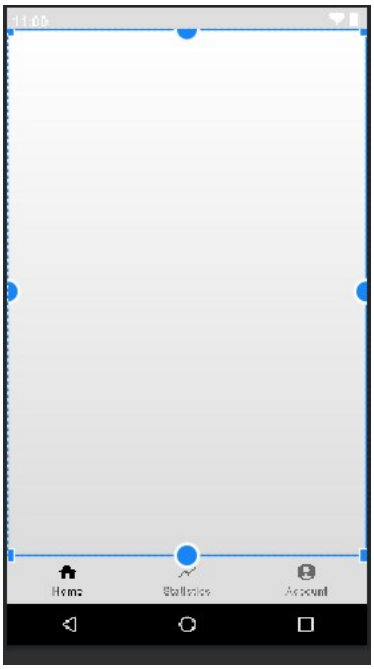
Slika 4.1. Postavke aktivnosti za DispatcherActivity aktivnost

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_forgot_passowrd)

    auth = Firebase.auth

    var activityClass :Class<*>

    val prefs = getSharedPreferences(
        sharedPreferencesStorageKey,
        MODE_PRIVATE
    )

    if(auth.currentUser == null){
        var userEmail = prefs.getString(userEmailStorageKey,null)
        var userPassword = prefs.getString(userEmailStorageKey,null)
        activityClass = if(userEmail!=null && userPassword!=null){
            auth.signInWithEmailAndPassword(userEmail,userPassword)
            HomeActivity::class.java
        }else{
            LoginActivity::class.java
        }

    }
    else{
        activityClass = HomeActivity::class.java
    }

    startActivity(Intent( packageContext: this, activityClass))
    finish()
}
```

Slika 4.2. Prikaz onCreate metode u DispatcherActivity aktivosti

4.1.2. Home aktivnost

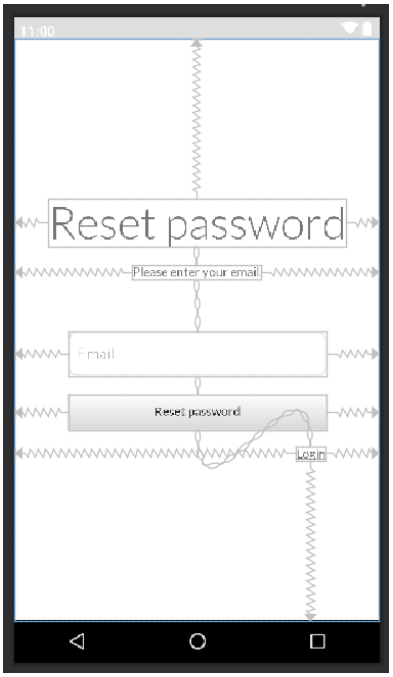
Home aktivnost služi za objedinjavanje svih fragmenata u aplikaciji. Sastoji se od prostora za fragmente i od prostora za navigacijsku traku, odnosno *BottomNavigationView* elementa u XML datoteci. (slika 4.3.)



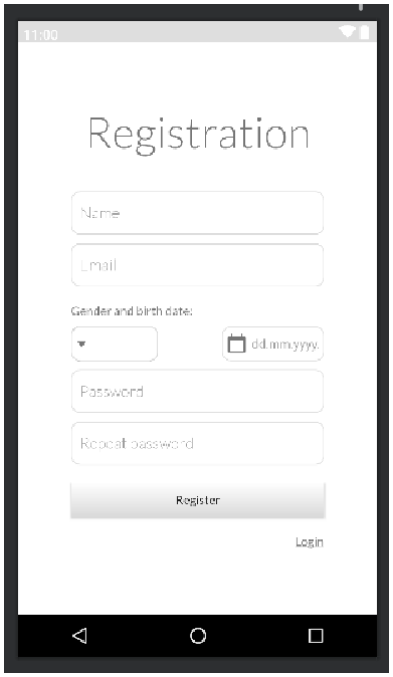
Slika 4.3. Predložak HomeActivity aktivnosti

4.2. Autentikacija

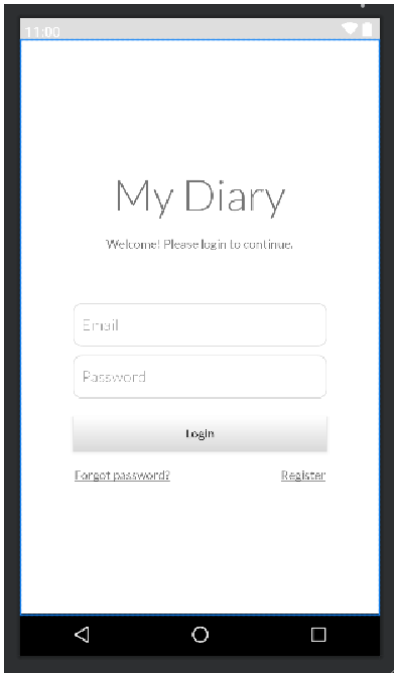
Pod autentikacijom obrađeni su svi dijelovi koji služe da se korisnik može upisati u bazu podataka, potvrditi svoj identitet te pristupiti svojim podacima.



Slika 4.4. Predložak za ponovno postavljanje lozinke



Slika 4.5. Predložak za registraciju



Slika 4.6. Predložak za prijavu

4.2.1. Registracija

Aktivnost zadužena za registraciju naziva se *RegistrationActivity*. XML datoteka na zaslonu za registraciju sastoji se od četiri *EditText-a*, jednog *Spinner-a* za odabir spola, jednog elementa za odabir datuma rođenja (*DatePicker*), jednog *TextView* elementa za naslov, te dvaju dugmadi – jedan za potvrdu registracije, a drugi za prijelaz na zaslon za prijavu. Predložak opisanog se može vidjeti na slici 4.5.

```
<EditText
    android:id="@+id/nameRegistrationET"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_marginHorizontal="60dp"
    android:layout_marginBottom="10dp"
    android:background="@drawable/custom_input"
    android:fontFamily="@font/lato_light"
    android:hint="Name"
    android:inputType="textPersonName"
    android:paddingStart="10dp"
    app:layout_constraintBottom_toTopOf="@+id/emailRegistrationET"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/titleRegistrationTV" />
```

Slika 4.7. Predložak EditText XML elementa

Na slici 4.7.. može se vidjeti prikaz jednog XML elementa. Konkretno to je *EditText* element koji služi za unos imena korisnika – to se može prepoznati po dodijeljenom ID-u (@+id/nameRegistrationET), gdje *name* predstavlja naziv polja, *Registration* predstavlja zaslon dok *ET* označava naziv elementa. To je konvencija koja se proteže kroz svaki zaslon u aplikaciji. Glavna metoda koju sadrži *RegistrationActivity* klasa je *registerSubmit*, može se vidjeti na slici 4.8.. Navedena metoda se poziva kad korisnik stisne *Register* dugme. Metoda prvo provjerava jesu li uneseni podaci ispravni, ako nisu, korisniku na to ukazuje te zahtjeva unošenje ispravnih podataka. U slučaju da su uneseni podaci ispravni, slijedi stvaranje instance *User* modela, zatim spremanje u bazu (slika 4.9.).

```
private fun registerSubmit(){
    var name = nameET.text.toString()
    var email = emailET.text.toString().trim()
    var date = dateHelper.convertToStringFormat(calendar.time)
    var gender = spinner.selectedItem.toString()
    var passwordFirst = passwordET.text.toString().trim()
    var passwordSecond = passwordRepeatET.text.toString().trim()

    if(name.isEmpty()){
        nameET.setError("Name"+ isRequiredErrorMessage)
        nameET.requestFocus()
        return
    }
    if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        emailET.setError("Email"+ isInvalidErrorMessage)
        emailET.requestFocus()
        return
    }
```

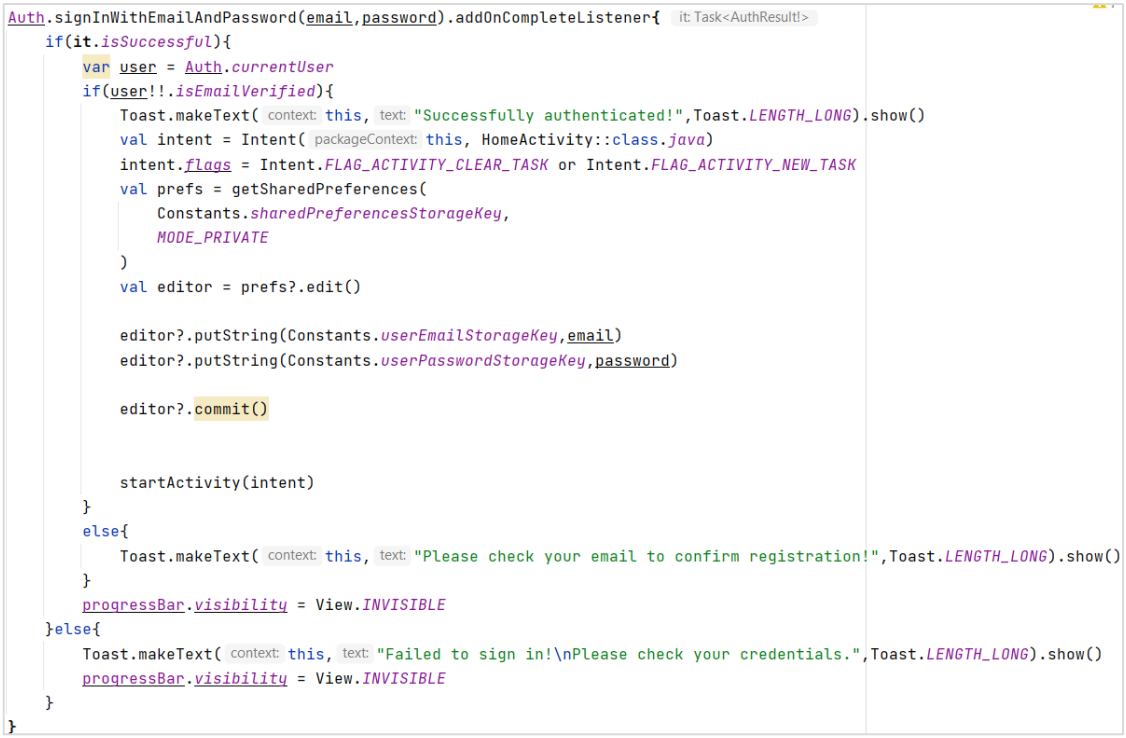
Slika 4.8. Prikaz registerSubmit metode

```
auth.createUserWithEmailAndPassword(email,passwordFirst)
    .addOnCompleteListener{ it: Task<AuthResult!>
        if(it.isSuccessful) {
            var user = User(name, email, gender, date, passwordFirst)
            database.getReference( path: "Users")
                .child(auth.currentUser!!.uid)
                .setValue(user)
                .addOnCompleteListener { it: Task<Void!>
                    if (it.isSuccessful) {
                        var currUser = auth.currentUser
                        currUser!!.sendEmailVerification()
                        Toast.makeText( context: this, registerSuccessfulMessage, Toast.LENGTH_LONG)
                            .show()
                        val intent = Intent( packageContext: this, LoginActivity::class.java)
                        startActivity(intent)
                    } else {
                        Toast.makeText( context: this, registerErrorMessage, Toast.LENGTH_SHORT)
                            .show()
                    }
                    progressBar.visibility = View.INVISIBLE
                }

        }else{
            Toast.makeText( context: this, registerErrorMessage, Toast.LENGTH_SHORT)
                .show()
            progressBar.visibility = View.INVISIBLE
        }
    }
```

Slika 4.9. Upisivanje korisnika u bazu

4.2.2. Prijava

Predložak zaslona za prijavu može se vidjeti na slici 4.6.. Aktivnost zadužena za prijavu u sustav naziva se *LoginActivity*. Najvažnija metoda kod prijave u sustav je *loginSubmit*, a u njoj se odvija postupak sličan kao i u *registerSubmit* metodi koja je prethodno opisana. Na slici 4.10. se može vidjeti dio koda koji se poziva u *loginSubmit* metodi i služi za prijavu u sustav.



Slika 4.10. Dio koda koji služi za prijavu korisnika u sustav

Ono što je bitno naglasiti je da nakon što je ustanovljeno da su podaci ispravni stvara se *Intent* koji služi da bi se prešlo sa postojeće aktivnosti na drugu (u ovom slučaju *HomeActivity*). Također na stvoreni *Intent* dodaju se zastave (engl. *flags*) kojima postižemo da se korisnik ne može vratiti na prethodnu aktivnost klikom na dugme *nazad*. Nakon stvaranja *Intenta,* uneseni email i lozinka spremaju se u dijeljene preferencije (engl. *shared preferences*) kako bi se moglo ustanoviti u daljnjim postupcima je li korisnik prijavljen.

4.2.3. Ponovno postavljanje lozinke

Ovaj zaslon služi za ponovno postavljanje lozinke kad korisnik ima poteškoću sa prijavom u sustav zbog zaboravljene lozinke. Glavna metoda u *ForgotPasswordActivity* aktivnosti je *submitResetPassword* metoda u kojoj se preko *Firebase* klase poziva metoda *sendPasswordResetEmail* nakon koje korisnik odlaskom na email ima mogućnost promjene lozinke.

```
private fun submitResetPassword() {
    var email = emailET.text.toString()

    if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        emailET.setError("Email"+ Constants.isInvalidErrorMessage)
        emailET.requestFocus()
        return
    }

    auth.sendPasswordResetEmail(email).addOnCompleteListener{it->

        if(it.isSuccessful){
            Toast.makeText( context: this, text: "Please check your email.",Toast.LENGTH_LONG).show()
            var intent = Intent( packageContext: this, LoginActivity::class.java)
            startActivity(intent)
        }else{
            emailET.text = null
            Toast.makeText( context: this, text: "Invalid email. Please check your credentials.",Toast.LENGTH_LONG).show()
        }
    }
}
```

Slika 4.11. Prikaz metode zadužene za ponovno postavljanje lozinke

4.3. Bilješke

U nadolazećem tekstu bit će opisana aktivnost za upisivanje nove bilješke koja ujedno služi za pregled i uređivanje iste. Zatim bit će opisan fragment koji služi za prikaz bilješki.

4.3.1. Stvaranje, pregled i uređivanje bilješke

Aktivnost koja objedinjuje sve funkcionalnosti navedene u podnaslovu naziva se *NoteUpsertActivity*. Sam naziv ove aktivnosti govori da ovo nije aktivnost za samo stvaranje bilješke, nego i za uređivanje iste.

Slika 4.12. Predložak NoteUpsert aktivnosti

Na slici 4.12. se može vidjet predložak *NoteUpsertActivity* aktivnosti. Aktivnost pruža mogućnost odabira datuma na kojeg bilješka odnosi, pritom valja naglasiti da se ne mogu odabrati datumi za koje već postoji bilješka i datumi koji su u budućnosti. Odabrani datum se prikaže u tekstualnom obliku iznad naslova bilješke. Korisnik unosi naslov i tekstualni sadržaj bilješke.

Dodatne mogućnosti koje se korisniku pružaju je odabir fotografije iz galerije te snimanje glasovne poruke. Glasovna poruka kao i fotografija se spremaju lokalno u poseban direktorij u memoriji Android uređaja, dok se u bazu upisuje samo informacija postoji li fotografija ili glasovna poruka.

Postoje dvije vrste modela koji se koriste pri manipulaciji podataka za bilješke. Na slici 4.13. se vidi model koji se šalje u bazu podataka, dok na slici 4.14. se vidi model u kojeg se transformira bilješka kad se dohvati s baze podataka. Ključna razlika između ova dva modela je da je datum u modelu koji se šalje u tekstualnom obliku tj. *String*, dok je u modelu koji se koristi za manipuliranje u Date tipu podatka.

```
data class NoteBM(
    val id:String,
    val date:String,
    val title:String,
    val content:String,
    val hasImage:Boolean,
    val hasVoiceRecording:Boolean,
    val moodRate : Int
)
```

Slika 4.13. NoteBM model

```
@Parcelize
data class NoteDto(
    val id:String,
    val date:Date,
    val title:String,
    val content:String,
    val hasImage:Boolean,
    val hasVoiceRecording:Boolean,
    val moodRate : Int,
):Parcelable
```

Slika 4.14. NoteDto model

Nadalje, važan dio ove aktivnosti je spremanje i dohvaćanje fotografija i glasovnih poruka. Na slici 4.15. može se vidjeti dio koda koji se pokreće kad se pritisne dugme za odabir fotografije. Kako bi korisnik pristupio datotekama iz vanjske (eksterne) memorije mora dati dozvolu aplikaciji. Ako korisnik ima dozvolu poziva se funkcija *pickImageFromGallery*, u suprotnom se poziva funkcija *requestPermissions* u kojoj je navedeno koje točno dozvole se zahtjevaju.

```
R.id.imagePickBtn ->{
    if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
        if(checkSelfPermission(android.Manifest.permission.READ_EXTERNAL_STORAGE) ==
            PackageManager.PERMISSION_DENIED){
            val permissions = arrayOf(android.Manifest.permission.READ_EXTERNAL_STORAGE)
            requestPermissions(permissions, PERMISSION_CODE)
        }
        else{
            pickImageFromGallery()
        }
    }
}
```

Slika 4.15. Dio koda koji se poziva pri kliku za odabir fotografije

```
private fun pickImageFromGallery() {
    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "image/*"
    startActivityForResult(intent, IMAGE_PICK_CODE)
}
```

Slika 4.16. Prikaz pickImageFromGallery metode

Nakon što se u *pickImageFromGallery* pozove *Intent* koji otvara galeriju, odabrana fotografija se dohvati u *onActivityResult* metodi te se postavlja na sva potrebna mjesta, te se zastava *hasImage* postavlja na istinu.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(resultCode== RESULT_OK && requestCode == IMAGE_PICK_CODE){
        imageSmallIV.visibility=View.VISIBLE
        imageSmallIV.setImageURI(data?.data)
        imageBigIV.setImageURI(data?.data)
        imageUri = data?.data!!
        hasImage = true
    }
}
```

Slika 4.17. Prikaz onActivityResult metode

Na slici 4.18. se vide tri važne metode vezane uz snimanje glasovnih poruka. Kako bi se započelo snimanje glasovne poruke, mora se prvo pozvati *startRecording* metoda u kojoj se pomoću *MediaRecorder* klase postavljaju sve postavke za snimanje poput izvora zvuka, izlaznog formata i odredišta. Zatim se pozivom metode *start* započinje snimanje zvuka. Snimanje se zaustavlja pozivom metode *stopRecording*. Metoda *getRecordingFilePath* služi za definiranje putanje gdje će se spremati zvučni zapis. Svaki zapis se sprema pod jedinstvenim nazivom, a to je ID bilješke sa pripadajućom ekstenzijom (.mp3).

```
private fun startRecording() {
    try {
        mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC)
        mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP)
        mediaRecorder.setOutputFile(getRecordingFilePath())
        mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB)
        mediaRecorder.prepare()
        mediaRecorder.start()

        Toast.makeText( context: this, text: "Recording started",Toast.LENGTH_SHORT).show()

    }catch(e:Exception){
        e.printStackTrace()
    }
}

private fun stopRecording() {
    mediaRecorder.stop()
    mediaRecorder.reset()

    Toast.makeText( context: this, text: "Recording stopped",Toast.LENGTH_SHORT).show()
}

private fun getRecordingFilePath():String{
    val contextWrapper = ContextWrapper(applicationContext)
    val musicFile = contextWrapper.getExternalFilesDir(Environment.DIRECTORY_MUSIC)
    val file = File(musicFile, child: dateId+".mp3")
    return file.path
}
```

Slika 4.18. Prikaz metoda zaduženih za snimanje glasove poruke

Metoda unutar *NoteUpsertActivity* aktivnost koja je zadužena za spremanje bilješke naziva se *saveNote*. Kad se pozove prethodno navedena metoda prvo se naprave provjere valjanosti unesenih podataka, a pod time se smatra provjera popunjenosti polja i ispravnost odabranog datuma bilješke (datum ne smije biti u budućnosti, dok je odabir već postojećeg datuma spriječen jer su odabrani datumi onemogućeni u *DatePicker* elementu).

Zatim, stvara se instanca modela *NoteBM*, sprema se slika u memoriju (ako je ista učitana tokom stvaranja bilješke) pomoću *FileHelper-a* (bit će detaljnije opisan u poglavlju 4.6.) zatim se instanca modela sprema u bazu podataka pod putanjom „Notes/{id_prijavljenog_korisnika}/{id_bilješke}“ te se preusmjerava korisnika na fragment za prikaz svih bilješki.

```
val note = NoteBM(dateId,date,title,content, hasImage, hasVoiceRecording,moodRate)

if(imageUri != Uri.EMPTY){
    val iStream = contentResolver.openInputStream(imageUri)
    val inputData :ByteArray? = fileHelper.getBytes(iStream)

    fileHelper.writeImageToFile(note.id,inputData)
}

database.getReference( path: "Notes").
        child(auth.currentUser!!.uid).
        child(dateId).
        setValue(note).addOnCompleteListener{ it: Task<Void!>
            if(it.isSuccessful){
                Toast.makeText( context: this,noteSavedMessage, Toast.LENGTH_SHORT).show()
                val intent = Intent( packageContext: this, HomeActivity::class.java)
                startActivity(intent)
            }
            else{

                Toast.makeText( context: this,somethingWentWrong, Toast.LENGTH_SHORT).show()
            }
            progressBar.visibility=View.GONE
}
```

Slika 4.19. Dio koda zadužen za spremanje bilješke u bazu podataka

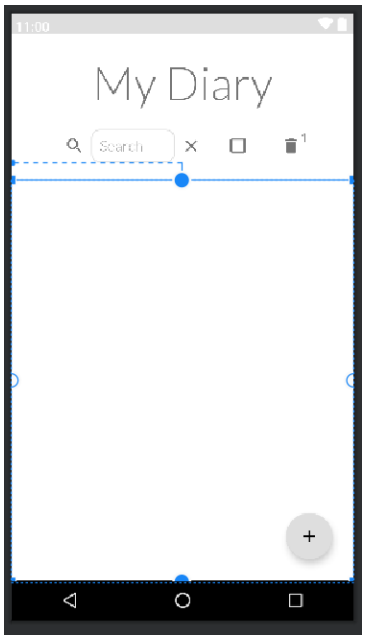
S obzirom da ova aktivnost služi i za uređivanje bilješke, na slici 4.20. se možda vidjeti dio zadužen za dohvaćanje podataka koji služe za uređivanje. Podatak iz *Intent-a* pod nazivom *upsertNote* je bilješka odnosno instanca NoteDto modela, čiji sadržaj je potrebno prikazati. Također dohvaćaju se već zauzeti datumi pod nazivom *disabledDays* te prvi unazad slobodan dan pod nazivom *freeDay* koji služi za inicijalno postavljanje datuma na bilješki. Zauzeti datumi i prvi slobodan dan se šalju iz *DiaryFragment* fragmenta pri prelasku na *NoteUpsertActivity*.

```
private fun fetchDataFromIntent() {
    val data = intent.getParcelableExtra<NoteDto>( name: "upsertNote")
    val disabledDaysString : ArrayList<String>? = intent.getStringArrayListExtra( name: "disabledDays")
    val freeDay : String? = intent.getStringExtra( name: "freeDay")
    if(freeDay != null){
    parsedFreeDay = dateHelper.getDateChartFormat(freeDay!!)}
    else{
        parsedFreeDay = Date() //value of this init is not important
    }

    disabledDays = arrayListOf()
    disabledDaysString?.forEach { it->
        val date = dateHelper.convertToDateFormat(it)
        val day = dateHelper.getDateDay(date)
        val month = dateHelper.getDateMonth(date)-1
        val year = dateHelper.getDateYear(date)
        val cal = Calendar.getInstance()
        cal.set(year,month,day)
        disabledDays?.add(cal)}

    if (data != null) {
        isUpsert = true
        upsertNote = data;
    } else {
        isUpsert = false
        upsertNote = NoteDto( id: "", Date(), title: "", content: "", hasImage: false, hasVoiceRecording: false, moodRate: 0)
    }
}
```

Slika 4.20. Prikaz metode namijenjene za dohvaćanje podataka bilješke za prikaz i uređivanje

4.3.2. Prikaz bilješki

Prikaz svih bilješki vezanih uz jednog korisnika prikazuju se na fragmentu *DiaryFragment*. Ovaj fragment osim prikaza svih bilješki omogućuje pretraživanje, označavanje te brisanje jedne ili više bilješki. Klikom na bilješku ili klikom na dugme za dodavanje nove bilješke preusmjerava se korisnika na *NoteUpsert* aktivnost.



Slika 4.21. Predložak MyDiaryFragment fragmenta

Dio koda zadužen za dohvaćanje bilješki prikazan je na slici 4.22.. S obzirom da podaci koji se dohvate s baze podataka razlikuje se za male nijanse od klasičnog JSON oblika, potrebno je vrijednost koju se dohvati, pretvoriti u JSON oblik pa tek onda pretvoriti u *NoteBM* model. Iz dobivenog *NoteBM* modela podaci se pretvaraju u *NoteDto*, za transformaciju datuma koristi se *DateHelper* (detaljnije u poglavlju 4.6.). Pri dohvaćanju se također i spremaju zauzeti datumi u polje *stringova disabledDays*. Bilješke su sortirane od najnovije prema najstarijima.

```
private fun fetchNotes(){
    this.fetchedNotes = mutableListOf()
    this.disabledDays = arrayListOf()
    progressBar.visibility = View.VISIBLE
    database.getReference( path: "Notes").child(auth.currentUser!!.uid).get().addOnCompleteListener {
        if(it.isSuccessful){
            it.result.children.forEach { data ->
                val toJson = gson.toJson(data.value)
                val note = gson.fromJson(toJson, NoteBM::class.java)
                val parsedDate = dateHelper.convertToDateFormat(note.date)
                this.fetchedNotes.add(NoteDto(note.id,parsedDate,note.title,note.content,note.hasImage,
                    note.hasVoiceRecording, note.moodRate))
                allDays.add(dateHelper.getDateChartFormat(parsedDate))
                disabledDays.add(note.date)
            }

            this.fetchedNotes.sortByDescending { it.date }
            notesRV.adapter!!.notifyDataSetChanged()
            progressBar.visibility = View.GONE
        }else{
            progressBar.visibility = View.GONE
        }
    }
}
```

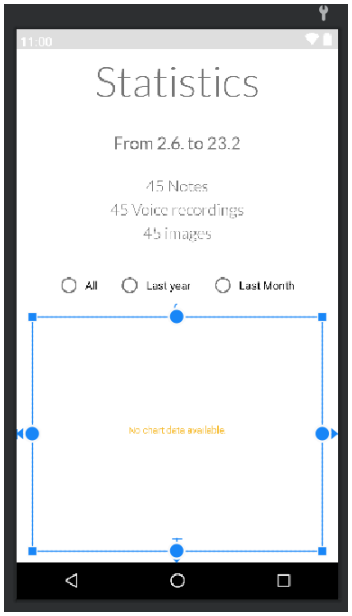
Slika 4.22. Dio koda zadužen za dohvaćanje bilješki

4.4. Statistički dio

Statistički dio je primarno zadužen kako bi korisnik dobio informacije o svom djelovanju u određenom vremenskom periodu. Od dobivenih informacija koje pruža statistički dio, korisnik može zaključiti u kolikoj mu se mjeri mijenja raspoloženje kroz vrijeme, koji je ukupan broj bilješki upisao, dodao fotografija i glasovnih poruka.

4.4.1. Prikaz statističkih podataka

Na slici 4.23. se može vidjeti predložak *StatisticsFragment* fragmenta namijenjenog za prikaz statističkih podataka.



Slika 4.23. Predložak StatistcsFragment fragmenta

Najvažnija metoda je *fetchData* u kojoj se dohvaćaju sve bilješke, sortiraju po datumu, filtriraju po odabranom vremenskom intervalu te pri prolasku kroz polje bilješki vrši se prebrojavanje koliko ukupno ima fotografija i glasovnih poruka.

```
barEntry.add(BarEntry(i.toFloat(), note.moodRate.toFloat()))
```

Slika 4.24. Dodavanje BarEntry instance u polje

Pri svakoj iteraciji kroz polje bilješki dodaje se instanca *BarEntry* klase, koja predstavlja x i y koordinatu, u polje *barEntry* (slika 4.24.).

Dobiveno polje *BarEntry* klase se sprema u instancu *BarDataSet* klase i pritom postavlja naslov grafa, boje stupaca u grafu te veličina fonta. Zatim se *BarDataSet* predaje konstruktoru *BarData* klase koji se dalje prosljeđuje direktno na XML element. Prethodno opisano može se vidjeti na slici 4.25..

```
val barDataSet = BarDataSet(barEntry,  label: "Mood chart")
barDataSet.setColors(colors)
barDataSet.valueTextSize = 15f
barDataSet.setDrawValues(false)

val barData = BarData(barDataSet)

barChart.animateY( durationMillis: 1100)
barChart.data = barData
barChart.isClickable = false
barChart.isContextClickable = false
barChart.description.isEnabled = false
barChart.legend.isEnabled = false
barChart.xAxis.valueFormatter = IndexAxisValueFormatter(datesModel)
barChart.xAxis.position = XAxis.XAxisPosition.BOTTOM
barChart.xAxis.setLabelCount(datesModel.size)
barChart.xAxis.setCenterAxisLabels(true)
barChart.setFitBars(true)
barChart.xAxis.textSize = 12f
barChart.xAxis.setAvoidFirstLastClipping(true)
barChart.xAxis.setCenterAxisLabels(true)

group.visibility = View.VISIBLE
progressBar.visibility = View.GONE
```

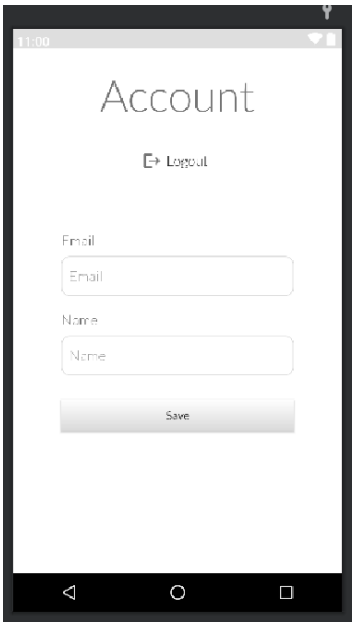
Slika 4.25. Definiranje podataka i postavki za prikaz grafa

4.5. Osobni podaci

Pri registraciji svaki korisnik ostavi određene podatke o sebi, to su: ime, email, spol, datum rođenja te lozinka. Ovaj dio aplikacije služi za pregled i izmjenu tih podataka.

4.5.1. Prikaz i uređivanje osobnih podataka

Na slici 4.26. je prikazan predložak *AccountFragment* fragmenta. Osim uređivanja osobnih podataka ovaj zaslon korisniku pruža opciju odjave iz sustava. Klikom na dugme za odjavu iz sustava poziva se kod koji je prikazan na slici 4.27.. Korisnik biva odjavljen iz sustava pozivom metode *signOut* te nakon toga se preusmjerava korisnika na zaslon za prijavu.



Slika 4.26. Predložak AccountFragment fragmenta

```
R.id.logoutBtn ->{
    auth.signOut()

    val intent = Intent(activity, LoginActivity::class.java)
    activity?.startActivity(intent)
    activity?.finish()
}
```

Slika 4.27. Dio koda namijenjen za odjavu iz sustava

4.6. Pomoćne klase

U ovom poglavlju su prikazane i opisane klase čije funkcionalnosti se koriste na više mjesta u kodu te baš iz tog razloga su izdvojene. *FileHelper* i *DateHelper* su klase koje su koje su ubrizgane u ostale klase pomoću Hilt biblioteke.

```
class DiaryFragment : Fragment(R.layout.fragment_diary), View.OnClickListener, NoteAdapter.OnItemClickListener{
    @Inject
    lateinit var dateHelper: DateHelper
    @Inject
    lateinit var fileHelper: FileHelper
```

Slika 4.28. Primjer ubrizgavanja klasa

Primjer ubrizgavanja klasa može se vidjeti na slici 4.28.. Bitno je naglasiti da iznad klase koja u sebe ubrizgava ovisnosti preko Hilt biblioteke treba sadržavati *@AndroidEntryPoint* oznaku, a klase koje se ubrizgavaju iznad sebe trebaju sadržavati *@Inject* oznaku.

4.6.1. Upravljanje datotekama

S obzirom da se u ovom radu upravlja fotografijama i glasovnim porukama, njihovo spremanje te dohvaćanje unificirano je u *FileHelper* klasi. Klasa *FileHelper* se ubrizgava u sve klase u kojima se vrši dohvaćanje ili stvaranje datoteka.

```
class FileHelper @Inject constructor(@ApplicationContext val context: Context) {
    private fun deleteFileFromExternalStorage(filename: String,extension: String,environment: String):Boolean{...}
    private fun writeToFile(fileName : String,data:ByteArray?, extension:String, environment: String){...}
    fun getBytes(inputStream: InputStream?): ByteArray? {...}
    fun writeImageToFile(fileName : String,data:ByteArray?){...}
    fun deleteImageFile(filename: String):Boolean{...}
    fun deleteVoiceRecordingFile(filename: String):Boolean{...}
}
```

Slika 4.29. Prikaz FileHepler klase

Na slici 4.29. prikazana je klasa *FileHelper* te potpis svih metoda koje ona sadrži. Prve dvije metode su privatne jer se koriste unutar metoda koje su izložene javno.

4.6.2. Upravljanje datumima

Sve potrebne transformacije datuma su delegirane na ovu klasu. Primjerice, korištenje ove klase je nužno kod pretvaranja datuma iz *String* u *Date* format kad se dohvaća s baze podataka. Prikaz *DateHelper* klase s potpisom metoda može se vidjeti na slici 4.30..

```
class DateHelper @Inject constructor() {
    private val sdf: SimpleDateFormat = SimpleDateFormat(DATE_FORMAT)
    fun convertToDateFormat(date: String):Date{...}
    fun convertToStringFormat(date:Date):String{...}
    fun getDateDay(date:Date): Int {...}
    fun getDateMonth(date:Date): Int {...}
    fun getDateYear(date:Date): Int {...}
    fun getDateChartFormat(date:Date): String {...}
    fun getDateChartFormat(date:String): Date {...}
}
```

Slika 4.30. Prikaz DateHelper klase

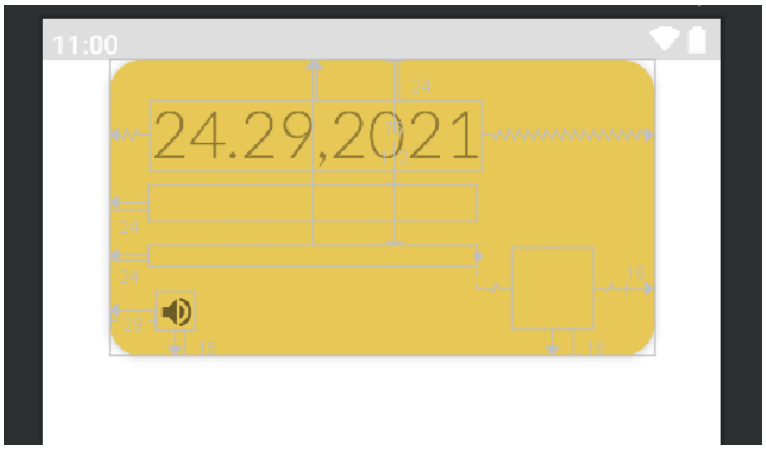
4.6.3. Adapter za bilješke

NoteAdapter klasa služi isključivo za prikazivanje podataka u *RecyclerView* elementu u *DiaryFragmentu*. Adapter u konstruktoru prima bilješke, odabrane bilješke, *OnItemClickListener* element te kontekst. *SelectedNotes* su bilješke koje su označene od strane korisnika, te spremne za brisanje. Zastavica *isSelected* naznačuje je li adapter u načinu rada za označivanje bilješki.

```
class NoteAdapter(
    var notes: MutableList<NoteDto>,
    var selectedNotes:MutableList<NoteDto>,
    val listener : OnItemClickListener,
    val context : Context,
) : RecyclerView.Adapter<NoteAdapter.NoteViewHolder>() {
    var isSelected = false
    inner class NoteViewHolder(itemView:View) : RecyclerView.ViewHolder(itemView), View.OnClickListener, View.OnLongClickListener{...}
    interface  OnItemClickListener {...}
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NoteViewHolder {...}
    override fun getItemCount(): Int {...}
    override fun onBindViewHolder(holder: NoteViewHolder, position: Int) {...}
}
```

Slika 4.31. Prikaz NoteAdapter klase

U *NoteViewHolder* unutarnjoj klasi se dohvaćaju svi XML elementi vezani uz jednu bilješku. Sučelje *OnItemClickListener* služi kako bi ga *DiaryFragment* mogao naslijediti te definirati ponašanja za *onClick* i *onLongClick* metode. *onCreateViewHolder* povezuje XML dokument sa *NoteAdapter* klasom. Metoda *getItemCount* vraća ukupan broj bilješki u adapteru. *onBindViewHolder* metoda služi za postavljanje podataka na pojedinu bilješku. Na slici 4.32. je prikazan jedan element *RecyclerView*-a.



Slika 4.32. Predložak za jedan element RecyclerView-a

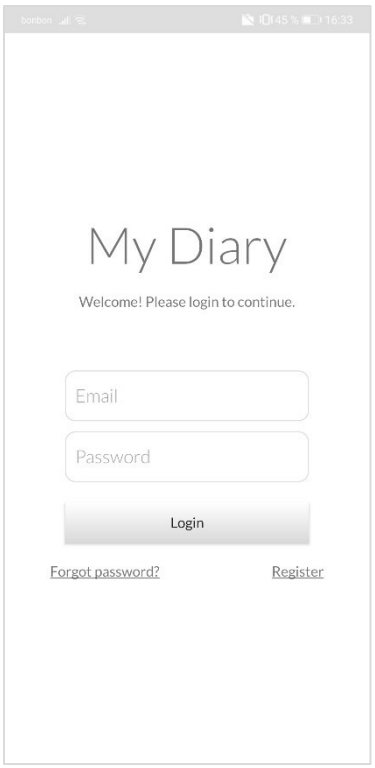
5. PRIKAZ ANDROID APLIKACIJE OSOBNI DNEVNIK

Ovo poglavlje služi za vizualni prikaz aplikacije i realizaciju prethodno opisanih stavki. Nadalje u tekstu pojedina slika će biti ukratko opisana.

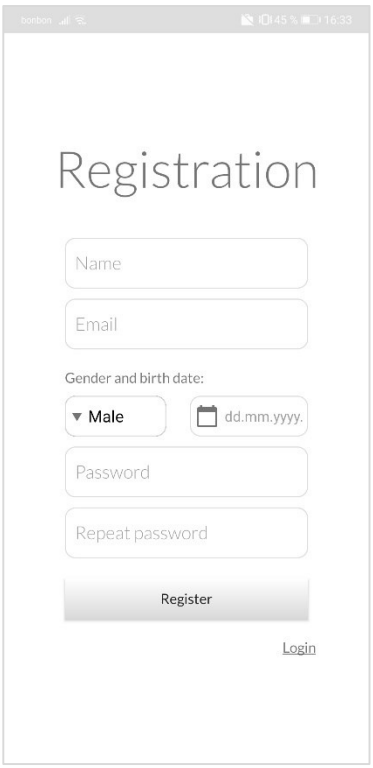
Na slici 5.1. prikazan je zaslon za prijavu u sustav koji se sastoji od polja za unos email-a i zaporke. To je ujedno i zaslon na kojeg se korisnika preusmjerava kad nije prijavljen u sustav. Na slici 5.2. prikazan je zaslon za registraciju. Kad se ispune sva polja i pritisne dugme za registraciju, ista se mora potvrditi preko email-a.

Na slici 5.3. prikazan je zaslon za ponovno postavljanje zaporke. Unosi se email na kojeg se šalje zahtjev za izmjenu zaporke. Slika 5.4. prikazuje zaslon za prijavu s unesenim podacima. Na slici 5.5. je prikazan zaslon za pregled svih bilješki, dok je na slici 5.6. prikazan primjer pretraživanja bilješki po unesenom tekstu u tražilici. Na slikama 5.7., 5.8. i 5.9. prikazano je označavanje bilješki. Slika 5.10. prikazuje pregled postojeće bilješke. Na slici 5.11. prikazano je odabir datuma na kojeg se bilješka odnosi. Važno je uočiti da su zauzeti datumi onemogućeni za odabir.

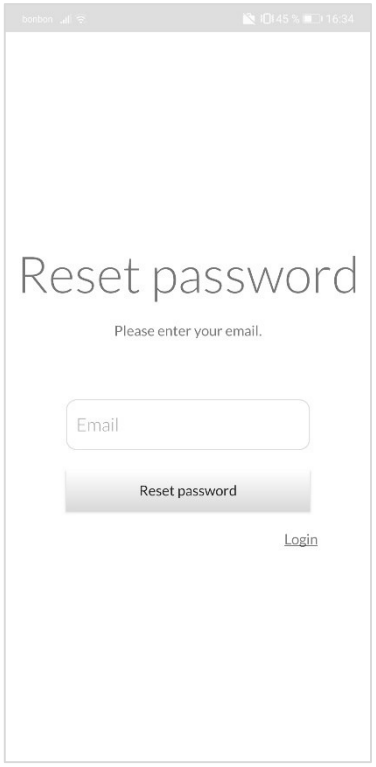
Slike 5.12. i 5.13. prikazuju *dialog* za ocjenjivanje raspoloženja koji se javlja kad se pritisne dugme za spremanje bilješke. Na slikama 5.14., 5.15. i 5.16. se mogu vidjeti zasloni zaduženi za statistiku. Prikazani su u tri različita stanja: odabir svih bilješki, bilješke u posljednjih godinu dana te posljednjih mjesec dana. Slika 5.17. prikazuje zaslon s osobnim podacima. Korisniku je pružena mogućnost uređivanja email-a i imena.



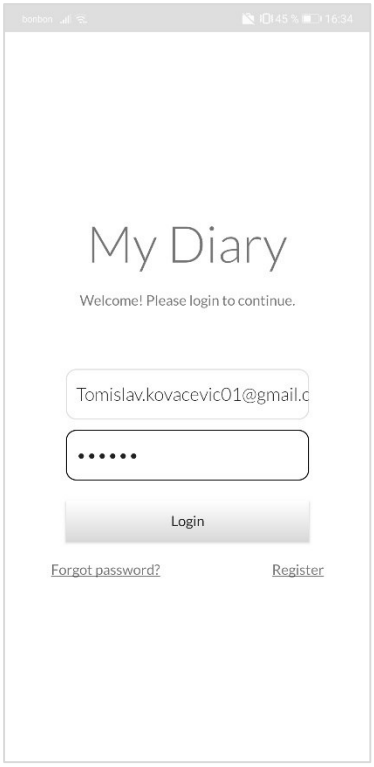
Slika 5.1. Prikaz zaslona za prijavu u sustav



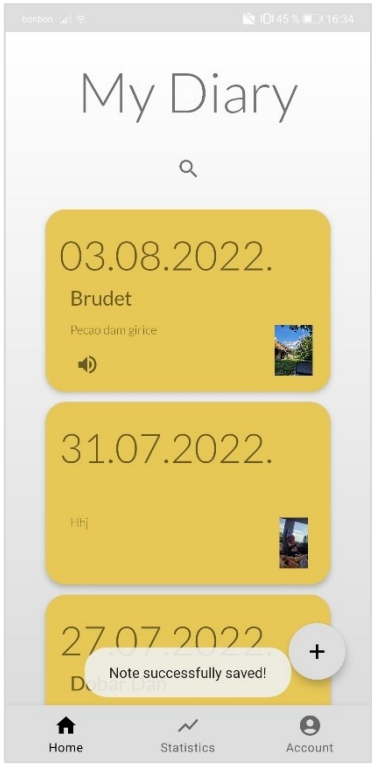
Slika 5.2. Prikaz zaslona za registraciju



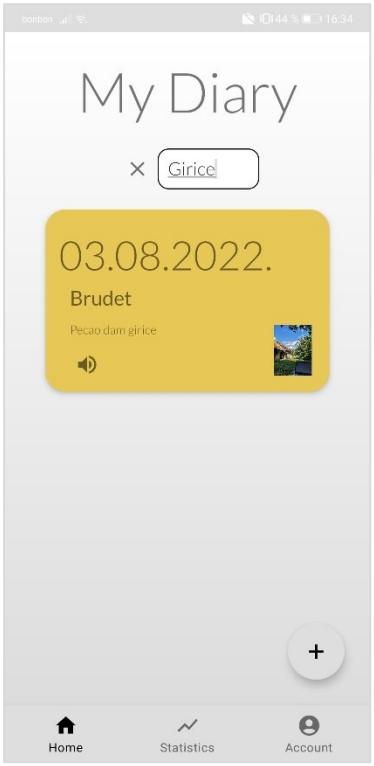
Slika 5.3. Prikaz zaslona za ponovno postavljanje lozinke



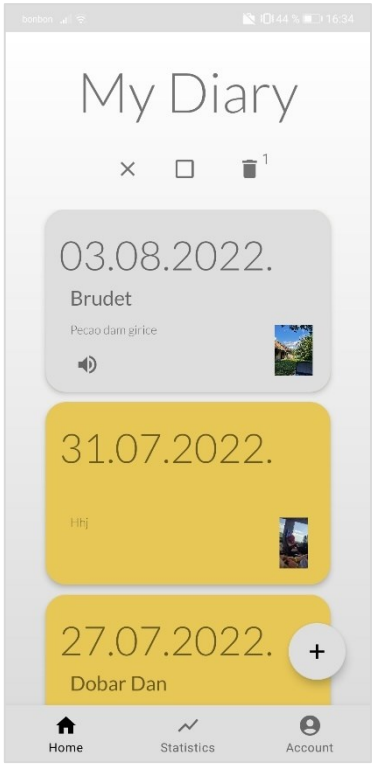
Slika 5.4. Prikaz zaslona za prijavu u sustav sa popunjenim podacima



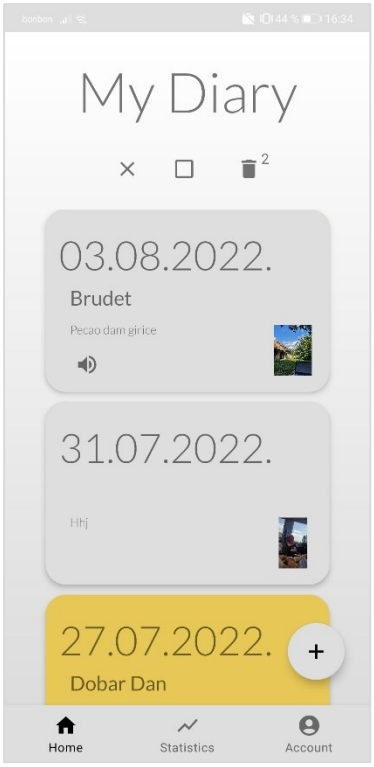
Slika 5.5. Prikaz zaslona za pregled svih bilješki



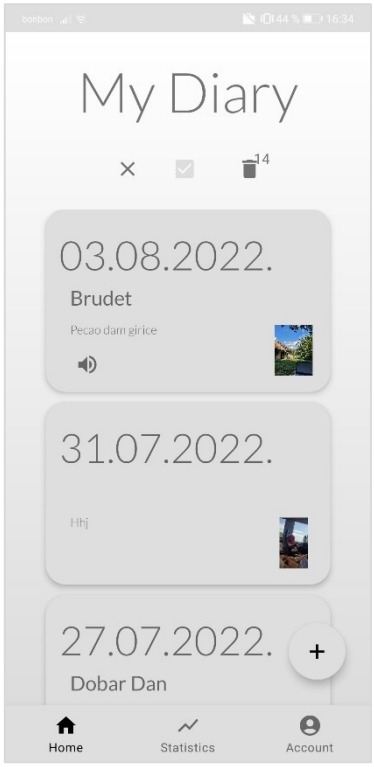
Slika 5.6. Primjer pretraživanja bilješki



Slika 5.7. Primjer označivanja bilješke



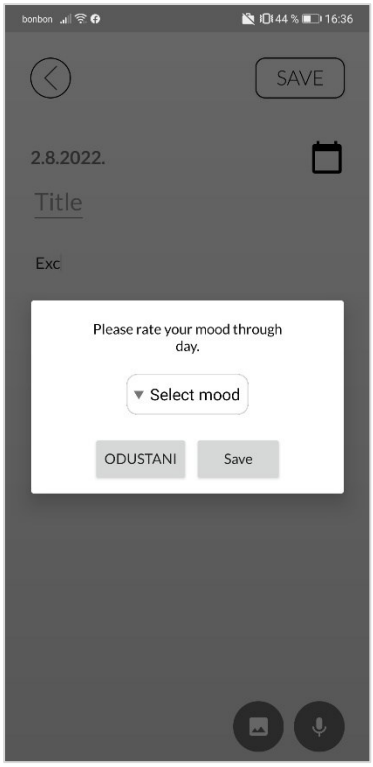
Slika 5.8. Prikaz označivanja više bilješki



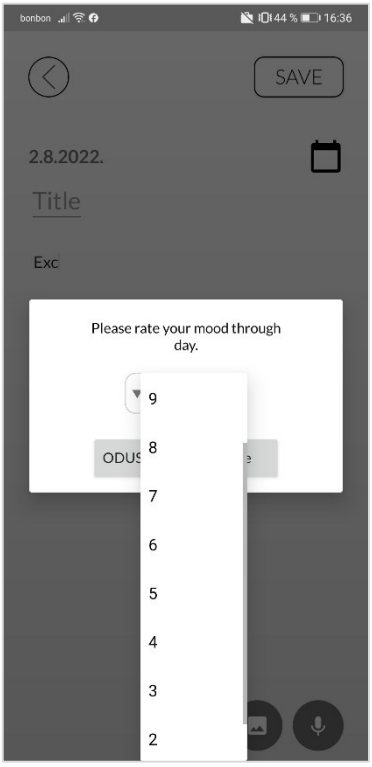
Slika 5.9. Primjer označivanja svih bilješki

Slika 5.10. Primjer otvaranja postojeće bilješke

Slika 5.11. Primjer odabira datuma (zauzeti su onemogućeni)

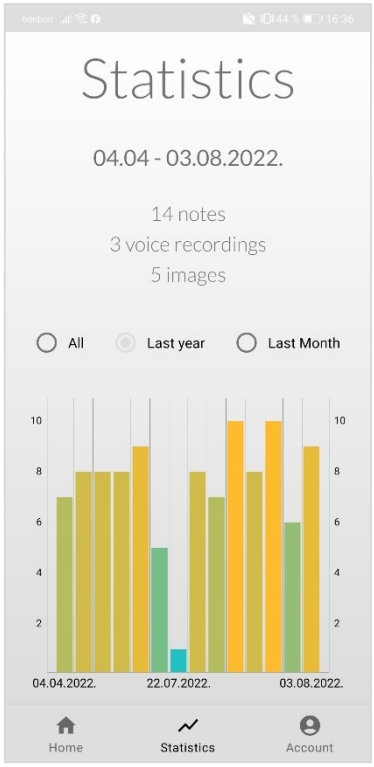


Slika 5.12. Dialog za odabir raspoloženja



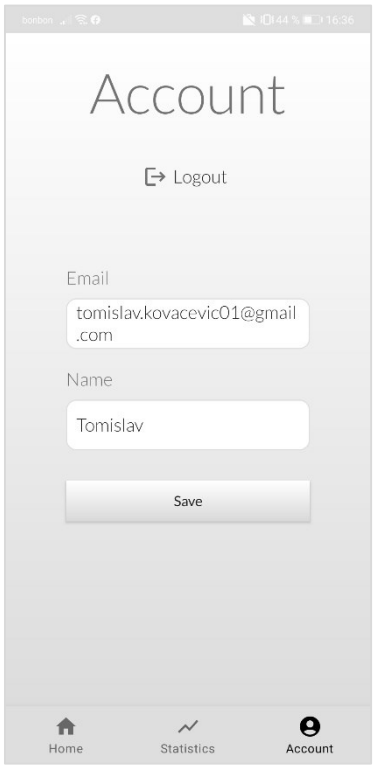
Slika 5.13. Dialog za odabir raspoloženja s prikazom ocjena

Slika 5.14. Prikaz zaslona za statistiku (sve bilješke)



Slika 5.15. Prikaz zaslona za statistiku (u godini)

Slika 5.16. Prikaz zaslona za statistiku (u mjesecu)



Slika 5.17. Prikaz zaslona sa osobnim podacima

6. ZAKLJUČAK

Cilj ovog rada nije samo izgraditi mobilnu aplikaciju, nego napraviti proizvod koji ima svrhu na tržištu i od kojeg će korisnici uistinu imati koristi. Ono što nudi ova aplikacija naizgled se može činiti beznačajno, no što je zaista cilj ove aplikacije je da se korisnik pisanjem svojih bilješki može bolje pozicionirati u vremenu u kojem se trenutno nalazi te time može biti prisutniji u trenutku. Korisnik se može vratiti na prethodne bilješke te analizirati svoje misli, ciljeve, potrebe te iz njih naučiti nešto za buduće akcije u životu. Osim tekstualne analize svojih promišljanja, može pratiti i mijene svojih subjektivnih osjećaja tokom određenog vremenskog perioda.

Ova aplikacija ima prostora za daljnji rast i razvoj. Neke od potencijalnih nadogradnji su primjerice spremanje fotografija te glasovnih poruka na internetski servis, time bi se postiglo da se ti podaci ne vežu samo lokalno uz uređaj, a baza bi se mogla iskoristiti iz za npr. web aplikaciju. Također, ono što bi bilo korisno dodati je izvoz u *PDF* format kako bi korisnik podatke mogao spremati gdje mu odgovara.

Aplikacija je funkcionalna, testirana te pokriva sve što je navedeno u zadatku završnog rada.

LITERATURA

[1] ‘Meet Android Studio’, dostupno na:
https://developer.android.com/studio/intro
pristupljeno 4. kolovoza 2022.

[2] ‘“FAQ - IntelliJ Open-Source Project - Confluence”, dostupno na:
www.jetbrains.org.
pristupljeno 4. kolovoza 2022.

[3] A modern programming language that makes developers happier , dostupno na:
‘https://kotlinlang.org/’.
pristupljeno: 4. kolovoza 2022.

[4] ‘XML ESSENTIALS’, dostupno na:
https://www.w3.org/standards/xml/
pristupljeno: 4. kolovoza 2022.

[5] ‘Firebase’, dostupno na:
https://docs.flutter.dev/development/data-and-backend/firebase
pristupljeno: 4. kolovoza 2022.

[6] ‘Firebase Realtime Database’, dostupno na:
https://firebase.google.com/docs/database
pristupljeno: 4. kolovoza 2022.

[7] ‘Dependency injection with Hilt’, dostupno na:
https://developer.android.com/training/dependency-injection/hilt-android
pristupljeno: 4. kolovoza 2022.

SAŽETAK

Ovaj završni rad prikazuje i opisuje tehnologije koje su korištene za izradu Android aplikacije koja ima funkcionalnosti osobnog dnevnika. Aplikacija korisniku omogućuje zapisivanje poruka, fotografija te zvučnih zapisa koje su objedinjene jednom bilješkom. Osim zapisa bilješki aplikacija ima mogućnosti prikaza izvještaja na temelju unesenih podataka. U statističkom dijelu aplikacije se brojčano opisuje količina unesenih podataka te grafički prikazuje promjena raspoloženja korisnika kroz vrijeme.

Ključne riječi: android, aplikacija, osobni, dnevnik, firebase, statistka

ABSTRACT

Title: **PERSONAL DIARY**

This bachelor's thesis presents and describes the main elements that were used to create an Android application that has the functionality of a personal diary. The application allows user to record messages, photos and sound recordings that are combined into one note. The application also has the ability to display reports based on the entered data. In the statistical part of the application, the amount of entered data is numerically described and the change in the user's mood over time is graphically displayed.

Key words: android, application, personal, diary, firebase, statistics