Mobilna aplikacija za interakciju s Google virtualnim asistentom

Koščević, Klara

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

Mobilna aplikacija za interakciju s Google virtualnim asistentom

Završni rad br. 02/RAČ/2021

Klara Koščević

Bjelovar, Kolovoz 2021.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Koščević Klara** Datum: 15.07.2021. Matični broj: 001972

JMBAG: 0314019249

Kolegij: **PROGRAMIRANJE MOBILNIH APLIKACIJA**

Naslov rada (tema): **Mobilna aplikacija za interakciju s Google virtualnim asistentom**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Ante Javor, struč. spec. ing. comp.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag.ing.el., predsjednik**
2. **Ante Javor, struč.spec.ing.comp., mentor**
3. **Ivan Sekovanić, mag.ing.inf.et comm.techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 02/RAČ/2021

U radu je potrebno istražiti mogućnosti i implementirati segmente dostupnog programskog sučelja za Google virtualnog asistenta u nativnu Android aplikaciju. Aplikacija će podržati akcije za unos i pregled bilješki, promjenu postavki. Interakcija s aplikacijom biti će omogućena korištenjem glasovnih naredbi prema Google asistentu. Asistent može otvoriti aplikaciju, unijeti bilješku i pretražiti bilješke po pojmu te rezultate prezentirati kranjem korisniku.

Zadatak uručen: 15.07.2021.

Mentor: **Ante Javor, struč. spec. ing. comp.**

REPUBLIKA HRVATSKA 2 BJELOVAR VELEUČILIŠTE U BJELOVARU

Sadržaj

Tablica slika

1. UVOD

Svi novi mobilni uređaju imaju i instalirane desetke aplikacija. Koje točno aplikacije imate, ovisit će o operativnom sustavu. Operativni sustav s najviše dostupnih aplikacija je Android jer je najkorišteniji operativni sustav. Dobra stvar za Android mobitele je što kao predinstalirane aplikacije dolaze i Google aplikacije, poput Google virtualnog asistenta.

Google virtualni asistent omogućuje interakciju s aplikacijama na mobitelu. Mali broj ljudi koristi Google virtualni asistent, za razliku od Siri. Siri je virtualni asistent IOS operativnog sustava. Većina IOS korisnika koristi Siri za lakši rad s uređajem. Google virtualni asistent nema toliku popularnost jer se donedavno i slabo govorilo o Google virtualnom asistentu. Google virtualni asistent je lagan za korištenje jer radi s jednostavnim naredbama poput „Hey Google call mom“ te tako ubrza radnju. Osim poziva, Google virtualni asistent će umjesto vas poslati poruku, pustiti pjesmu, naručiti hranu ili kupiti karte za koncert. A samo je potrebno izreći ili napisati naredbu. Što se više služite Google virtualnim asistentom on postaje zanimljiviji. Svaki dan se pronalaze nove naredbe koje nekad ni ne ubrzaju radnju već otkriju nešto novo.

Svrha rada je izraditi android aplikaciju za stvaranje, uređivanje i pregled bilješki, koja vrši interakciju s Google virtualnim asistentom. Preko Google virtualnog asistenta je moguće otvarati određena korisnička sučelja aplikacije, pretraživati i stvarati bilješku. U radu će prvo biti opisan Android operativni sustav i korisnička podrška te Google virtualni asistent. Treće poglavlje opisuje što je obrada prirodnog jezika te gdje se koristi. Izgradnja baze podataka, izgradnja aplikacije, povezivanje aplikacije i baze podataka te implementacija interakcije s Google virtualnim asistentom će se obraditi u četvrtom poglavlju. Kao zadnje poglavlje nalazi se zaključak.

2. ANDROID I KORIŠTENA PROGRAMSKA PODRŠKA

Prilikom izrade android nativne aplikacije korišten je Android studio. Aplikacija je pisana u Kotlin programskom jeziku te je integrirana s Google virtualnim asistentom.

2.1 Android

Prema izvoru [1] Android je najčešće korišteni mobilni operativni sustav. Prvobitno je razvijen od strane Android Inc, no 2005. godine ga je kupio Google. Prvobitni razlog za razvoj je bio operativni sustav za digitalne fotoaparate, no promijenio se u operativni sustav za pametne mobitele. Do danas je izdano 24 verzije operativnog sustava.

Android je besplatan operativni sustav otvorenog koda. No Android uređaji dolaze s prethodno instaliranim vlasničkim sustavom koji sadrži osnovne Google aplikacije. Prema izvoru [2] Android operativni sustav se zapravo sastoji od mnogo softverskih komponenti koje su podijeljene u pet odjeljaka:

- Linux jezgra - upravlja izlaznim i ulaznim zahtjevima iz softvera
- biblioteke
- android izvršitelj (engl. *runtime*) - pruža ključnu komponentu Dalvik Virtual Machine. Dalvik VM pokreće aplikacije na android uređajima te omogućuje svakoj Android aplikaciji pokretanje vlastitog procesa.
- Programski okvir (engl. *framework*) - pruža usluge više razine aplikaciji koje pomažu razvojnim programerima
- Aplikacije.

Postoje 2 vrste mobilnih aplikacija, hibridne i nativne aplikacije. Razlika između hibridnih i nativnih aplikacija su za koji operativni sustav su namijenjene. Nativne aplikacije namijenjene su za jedan operativni sustav, primjerice Android. Za razliku od hibridnih koje su namijenjene za veći broj operativnih sustava. Također, razlika je i u cijeni, vremenu izrade, ali i u performansama. Hibridne jesu jeftinije i brže se izrade od nativnih. ali nativne imaju bolje performanse jer su rađene za samo jedan operativni sustav.

Prema izvoru [3] službena podrška za Kotlin je najavljena 2017. godine. Kotlin 2018. godine zamjenjuje JAVA programski jezik kao prvi izbor za izradu android nativnih aplikacija. Prema izvoru [4] neki od razloga za vodstvo Kotlina su da je potrebno manje linija koda i veća je čitljivost, ima jako veliku podršku i proširenja koja olakšavaju

kodiranje. Također, moguće ga je koristiti i u kombinaciji s Java-om. Jednostavan je za učenje, pogotovo Java programerima.

2.2 Android studio

Android studio je službeno integrirano razvojno okruženje za izradu Android aplikacija. Razvoj se je temeljio na InteliJ IDEA. Prema izvoru [5] Android studio nudi značajke za povećanje produktivnosti kao što su:

- brz emulator
- okruženje u kojem se može razvijati za sve Android uređaje
- predlošci koda i integracija GitHub-a
- razne mogućnosti testiranja aplikacije
- olakšana integracija s Google-om

Podržava programske jezike Java i Kotlin. Prilikom korištenja Java programskog jezika u Kotlinu ili obrnuto nudi mogućnost automatske promjene koda.

2.3 Google virtualni asistent

Prema izvoru [6] Google virtualni asistent je virtualni pomoćnik baziran na umjetnoj inteligenciji. Današnje mogućnosti Google asistenta su razne. Nudi naredbe, pretraživanje, kontrolu uređaja, upravljanje aplikacijama. Google virtualni asistent za razliku od prethodnika Google Now može voditi dvosmjerne razgovore. Prednosti Google virtualnog asistenta su također da se sve tekstualne naredbe mogu dati i glasovno. Kako bi pokrenuli glasovnu naredbu sve što je potrebno reći je “OK Google” ili “Hej Google”.

3. OBRADA PRIRODNOG JEZIKA

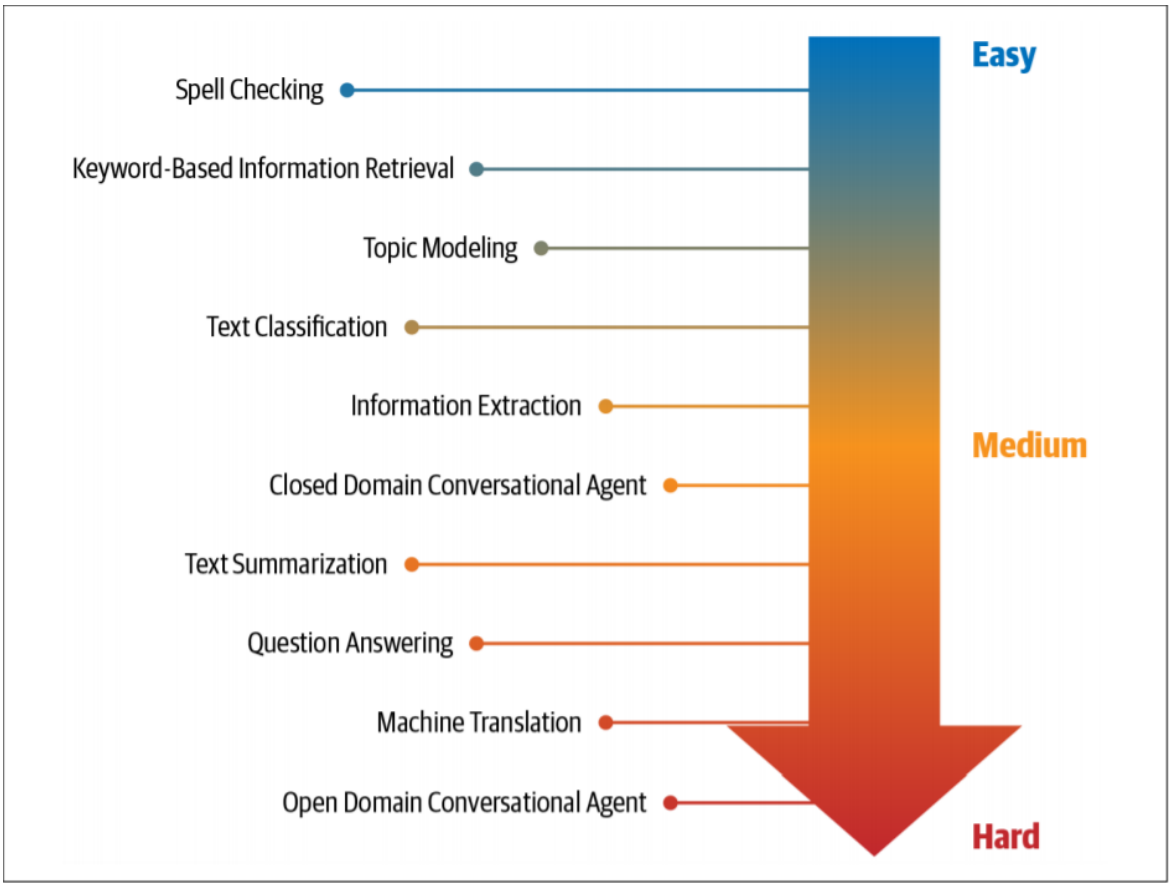
Prema izvoru [7] obrada prirodnog jezika (engl. *Natural language processing*, skraćeno NLP) je polje koje se nalazi na sjecištu informatike, lingvistike i umjetne inteligencije. Početak razvoja je bio 1950-ih godina. Od tada pa do posljednjeg desetljeća NLP domena je bila akademska zajednica i istraživački laboratoriji. Posljednje desetljeće NLP alati postaju široko dostupni i mnogo lakši za korištenje. Također, sve više organizacija poput Googlea, Microsofta te Amazona, ulažu u interaktivne proizvode gdje je vrlo potreban NLP. Održivosti NLP-a doprinosi to što se počelo razvijati i za jezike s manjim govornim područjem.

Prilikom davanja naredbe, tj. komunikacije sa Siri, Alexa ili Google virtualnim asistentom naredba je dana na prirodnom, ljudskom, jeziku. Računalo ne razumije prirodni jezik, ono razumije binarni te se tu koristi NLP. NLP se bavi analizom, modeliranjem i razumijevanjem prirodnog jezika za računala.

3.1 Zbirka zadataka obrade prirodnog jezika

Za lakšu izradu NLP aplikacija postoji niz temeljnih zadataka koji se često pojavljuju u NLP aplikacijama. Prema izvoru [7] sljedeći zadaci su najčešće korišteni te na slici 3.1 može se vidjeti zadatke poredane prema težini razvoja.

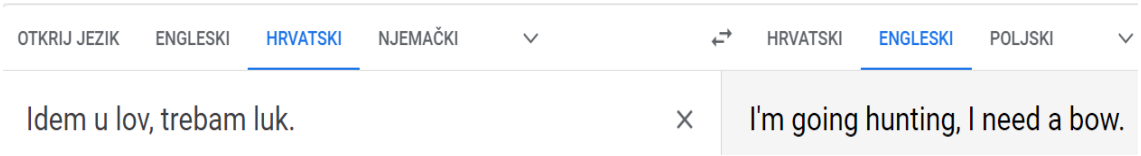
Zadatak modeliranje jezika – predviđa koje će se sljedeće riječi koristit u rečenici. Cilj je naučiti vjerojatnost pojave riječi. Korisno je za prepoznavanje govora, znakova, rukopisa, strojno prevođenje kao i ispravljanje pravopisa. **Zadatak klasifikacije teksta** – grupira tekst u poznati skup kategorija na temelju sadržaja. Jedan je od najpopularnijih zadataka. Koristi se od identifikacije neželjene e-pošte do analize osjećaja. **Zadatak izvlačenje informacija** – izvlači relevantne informacije iz teksta, poput imena ljudi spomenutih u objavi. **Zadatak povrat informacija** – pronalazi relevantne informacija za korisnički upit. Pretraživači poput Googlea i Binga najviše koriste ovaj zadatak. **Zadatak agent za razgovor** – izgrađuje sustav za dvosmjernu komunikaciju. Prije spomenuti Google virtualni asistent, Siri, Alexa i slično su uobičajeni primjeri ovog zadatka. **Zadatak sažimanje teksta** – stvara sažetke dokumenata bez izmjene temeljnog sadržaj i značenja dokumenta. **Zadatak odgovaranje na pitanje** – automatski odgovara na postavljena pitanja. **Zadatak strojno prevođenje** – prevodi tekst iz jednog jezika u drugi jezik. Svima poznati Google prevoditelj koristi upravo ovaj zadatak. **Zadatak modeliranje** – tema otkriva, tj. određuje temu velikoj zbirki dokumenata.



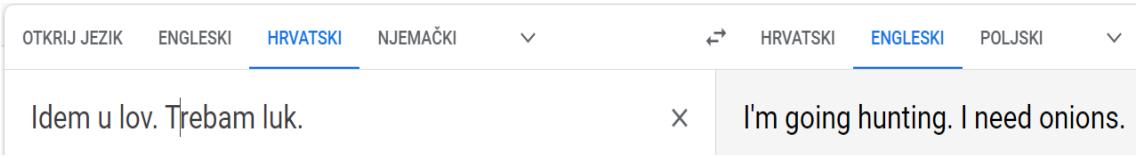
Slika 3.1 NLP najkorišteniji zadaci prema težini razvoja [7]

3.2 Problemi obrade prirodnog jezika

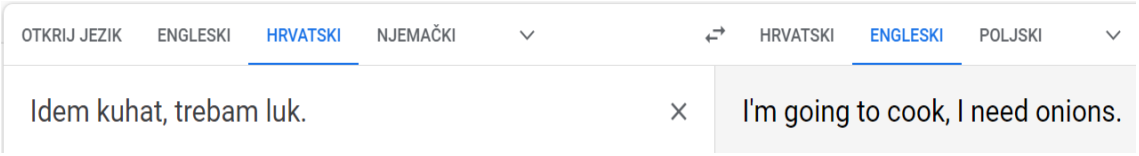
Većina jezika je dvosmislena odnosno jednu rečenicu možemo shvatiti na dva načina ovisno o kontekstu. Tako “Treba mi jedan luk”, ako govorimo o kuhanju “luk” predstavlja povrće. Dok kada govorimo o lovu “luk” predstavlja oružje. Dvosmislenost rečenice kao i opće znanje stvaraju problem za NLP. Na slikama 3.2, 3.3 i 3.4 može se vidjeti kako dvosmislenost, kontekst i opće znanje stvaraju problem za NLP. Čovjek bi znao da se na slici 3.3 ne radi o „onions” već o „bow“ dok NLP to ne shvaća. Jezik ne slijedi samo neki niz pravila, nego je i kreativan. Veliki je problem ne samo za NLP već i za umjetnu inteligenciju napraviti da strojevi razumiju kreativnost.



Slika 3.2 Google prevoditelj prepoznaje „luk" kao oružje



Slika 3.3 Google prevoditelj ne prepoznaje "luk" kao oružje



Slika 3.4 Google prevoditelj prepoznaje "luk" kao povrće

3.3 Pristup obradi prirodnog jezika

Prema izvoru [7] pristupi za rješavanje NLP problema se dijele u tri kategorije: heuristika, strojno i duboko učenje.

Heuristički pristup rješavanja problema temelji se na izgradnji pravila. Programeri zapravo moraju formulirati pravila koja bi se mogla ugraditi u program i pomoći pri rješavanju problema. Takav sustav obično zahtijeva resurse poput rječnika, koji su bili sastavljeni i digitalizirani. Tijekom vremena su izgrađene baze znanja koje pomažu pri rješavanju NLP problema, a posebno NLP sustava koji se temelji na pravilima.

NLP sustavi temeljeni na pravilima mogu uključivati i oblike informacija poput regularnih izraza (engl. *regular expression*, skraćeno regex) i gramatike bez konteksta (engl. *Context-free grammar*, skraćeno CFG). Regex je skup znakova ili uzorak koji se koristi za pronalaženje podnizova u tekstu. CFG se koristi za pronalaženje kompleksnijih hijerarhijskih informacija koje regex možda ne bi mogao pronaći.

Svaki pristup strojnog učenja za NLP može se opisati kao tri zajednička koraka: izdvajanje značajki iz teksta, korištenje prikaza značajki za učenje te vrednovanje i poboljšanje modela. NLP često koristi sljedeće metode strojnog učenja: Naive Bayes, stroj s vektorima potpore (engl. *Support vector machines*, skraćeno SVM), skriveni Markovljev model (engl. *Hidden Markov model* , skraćeno HMM) i uvjetno slučajno polje (engl. *Conditional random field* , skraćeno CRF).

Naive Bayes je algoritam za klasifikacijske zadatke koji se temelji na Bayesovom teoremu[1]. Naive Bayes se najčešće koristi kao početni algoritam za klasifikaciju teksta iz

[1] Opisuje vrijednost događaja na temelju prethodnog znanja o uvjetima koji bi mogli biti povezani s događajem [17]

razloga što je jednostavan za naučiti i brzo se izvodi. SVM uči granicu odlučivanja za odvajanje podataka u različite klase. HMM pokušava modelirati skrivena stanja iz prethodno generiranih podataka. HMM koristi Markovljevu pretpostavku da svako skriveno stanje ovisi o prethodnom stanju. CRF se koristi za sekvencijalne podatke, tj. izvršava klasifikaciju za svaki element u nizu.

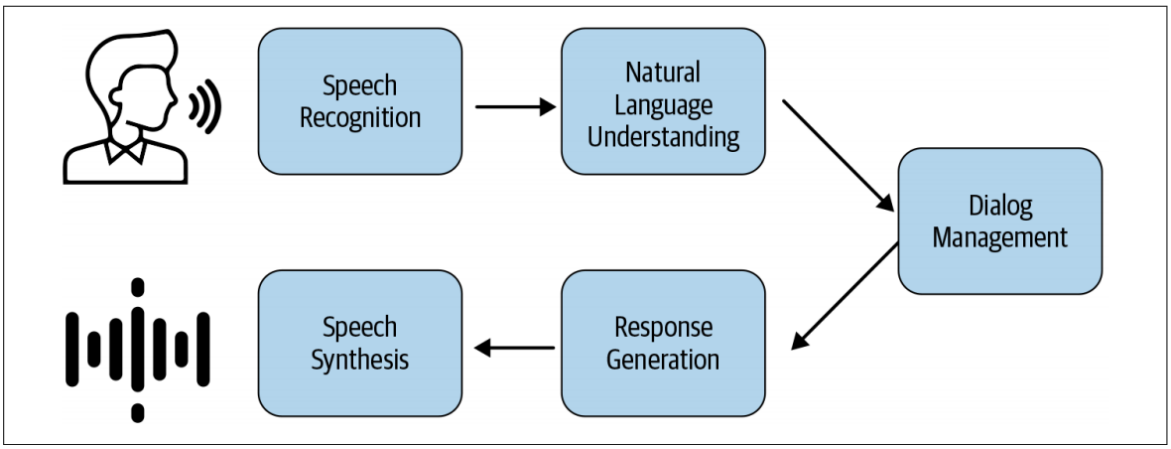
Duboko učenje se zasniva neuronskim mrežama[2]. Posljednjih nekoliko godina dolazi do porasta korištenja neuronskih mreža za bavljenje sa složenim i nestrukturiranim podacima. Iz tog razloga neuronske mreže su puno bolje za rješavanje jezičnih zadataka.

Ponavljajuće neuronske mreže (engl. *Recurrent neural network,* skraćeno RNN) osmišljene su za sekvencijalnu obradu i učenje. RNN se koristi za klasifikaciju teksta, prepoznavanje imenovanih entiteta, generiranje tekst i slično. Dugotrajno kratkotrajno pamćenje (engl. *Long short-term memory*, skraćeno LSTM) za razliku od RNN-a bolje radi s dužim tekstovima. LSTM odbacuje nevažni dio konteksta i sjeća se samo dijela koji je potreban za rješavanje zadatka. Konvolucijske neuronske mreže (engl. *Convolutional neural network*, skraćeno CNN) se u računalnom vidu koriste za klasifikaciju slika, prepoznavanje videa i slično. Upravo u klasifikaciji teksta CNN je postigao uspjeh. Svaka riječ u rečenici se može zamijeniti s vektorom te se na taj način tvori matrica ili dvodimenzionalni niz. Nakon što se stvori matrica, CNN je može modelirati. Najnoviji model dubokog učenja za NLP je transformator. Transformatori modeliraju tekstualni kontekst na način da gledaju sve riječi i predstavljaju svaku riječ s obzirom na njezin kontekst.

3.4 Agenti za razgovor

Agenti za razgovor poput Siri, Alexe i Google virtualnog asistenta koriste model interakcije koji se nalazi na slici 3.5. Prema izvoru [7] u nastavku slijedi ponešto o svakom koraku.

[2] Računalni sustavi inspirirani biološkim neuronskim mrežama



Slika 3.5. Tijek agenta za razgovor [7]

Prepoznavanje i sinteza govora su glavne komponente agenta za razgovor temeljenog na glasu. Prepoznavanje govora je pretvaranje govornih signala u foneme koji se prepisuju kao riječ. Sinteza pretvara tekstualni rezultat u govorni jezik.

Komponenta razumijevanje prirodnog jezika analizira odgovor korisnika pomoću sustava za razumijevanje prirodnog jezika. Ovu komponentu možemo podijeliti na nekoliko manjih podzadataka.

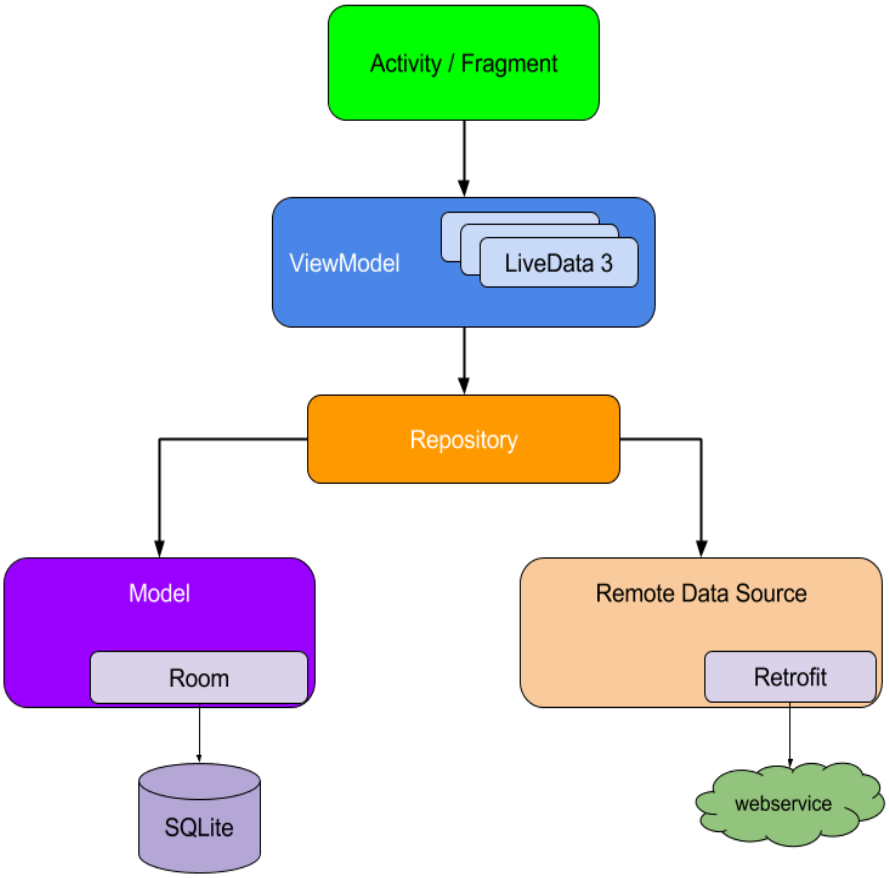
- Analiza osjećaja – analizira se raspoloženje korisnika
- Prepoznavanje imenovanog entiteta – identificiraju se svi važni entiteti koje korisnik spomene u odgovoru
- Koreferencija – saznajemo reference iz izdvojenih entiteta u prošlosti.

Nakon što se je dobivena korisna informacija, agenta zanima koja je namjera naredbe. Ponekad će se sustavom klasifikacije teksta klasificirati odgovor kao jednu od unaprijed definiranih namjera. Ukoliko ne postoji unaprijed definirana namjera agent će postaviti još dodatnih pitanja. Konačno agent će generirati odgovarajuću radnju za izvođenje. Radnja se temelji na semantičkoj interpretaciji namjere korisnika i dodatnih ulaza iz dijaloga razgovora. Agent može dohvatiti informacije iz baze podataka i generirati odgovore unaprijed pomoću definiranog predloška. Također može i generirati potpuno nove odgovore.

4. IMPLEMENTACIJA APLIKACIJE KORIŠTENJEM GOOGLE VIRTUALNOG ASISTENTA

Aplikacija razvijena za projekt služi za spremanje, pregled i uređivanje bilješki. Aplikacija ima svijetlu i tamnu temu, orijentacija može biti pejzaž ili portret te može biti na hrvatskom ili engleskom jeziku. Tema također može biti postavljena da ovisi o uređaju ili bateriji, a jezik se također može birati u postavkama. Preko Google virtualnog asistenta je moguće otvoriti prikaz svih bilješki ili postavke, pretražiti bilješke i spremiti novu bilješku.

Google ima „Upute za arhitekturu aplikacija“ (engl. *Guide to app architecture*) u kojima obuhvaća najbolju praksu i preporučenu arhitekturu. Prema „Uputama za arhitekturu aplikacija“ izgrađena je aplikacija. Klasama aktivnosti pristupamo operacijama nad bazama pomoću modela prikaza (engl. *viewModel*) koji pristupa klasi repozitorij koja tek onda pristupa operacijama nad bazom. Slika 4.1 prikazuje arhitekturu aplikacije.



Slika 4.1 Prikaz arhitekture iz „Upute za arhitekturu aplikacija“ [8]

4.1 Baza podataka

Za bazu podataka se koristi *Room*. „Room je biblioteka komponenti arhitekture *Jetpack* koja pojednostavljuje postavljanje baze podataka i pristup bazi podataka.“ [3] Prema izvoru [3] Room se sastoji od sučelja za programiranje aplikacija (engl. *application programming interface*, API), anotacije (engl. *annotations*) i prevoditelja (engl. *compiler*).

API sadrži klase za definiranje baze podataka i izgradnju njezine instance. Anotacije se koriste za označavanje stvari poput klase koje se pohranjuju u bazu podataka, koja klasa što predstavlja i koja klasa sadrži funkcije pristupa tablicama. Prevoditelj obrađuje označene klase i generira implementaciju baze podataka. Prilikom izgradnje baze podataka s Room bibliotekom postoje tri koraka. Prvo se mora definirati klasa koja sadrži entitet, zatim klasa koja predstavlja bazu podataka te na kraju pretvornik tipova (engl. *type converter*) da baza može rukovati podacima modela.

Prema izvoru [3] anotacija @Entity označava da se radi o klasi koja sadrži entitet. Klasa *NoteEntity* definira strukturu baze podataka, vidljivo u programskom kodu 4.1. U aplikaciji je potreban samo jedan entitet koji se zove *Note*. Anotacija @PrimaryKey označava da je taj stupac u bazi podataka primarni ključ. Podatak je primarni ključ kada je jedinstven za svaki unos. *Note* se sastoji od jedinstvenog identifikatora (ID) koji je tipa univerzalnog jedinstvenog identifikatora (engl. *universally unique identifier*, UUID), naziva i opisa bilješke tipa *string*[3] te datuma nastanka bilješke ili kada je zadnje uređivana, također tipa *string*.

Programski kod 4.1: Klasa NoteEntity

```
package com.kkoscevic.noteapp.database.entities

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity
data class Note(
    @PrimaryKey val id: UUID = UUID.randomUUID(),
    var title: String = ""
) {
    var noteText: String = ""
    var dateTime: String = ""
}
```

[3] *String* predstavlja nizove znakova [11]

Prema izvoru [3] anotacija @Database, vidljivo na programskom kodu 4.2, označava klasu koja predstavlja bazu podataka u aplikaciji. Anotacija zahtjeva dva parametra. Prvi parametar su klase koje se koriste pri stvaranju i upravljanju tablicama za bazu podataka. Drugi parametar je verzija baze. Prilikom prvog kreiranja baze verzija je jedan te se povećava svaki put kada se nešto mijenja u bazi. Treći korišteni parametar u kodu nije obavezan. Treći parametar označava hoće li se izvesti shema baze podataka u datoteku. Anotacija @TypeConverters označava koju klasu koristimo za pretvorbu tipova. U klasi *NoteDatabase* registriramo apstraktnu funkciju *noteDao* koja kao tip povrata ima klasu *NoteDao*.

Programski kod 4.2: Klasa NoteDatabase

```
package com.kkoscevic.noteapp.database

import androidx.room.Database
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import com.kkoscevic.noteapp.database.dao.NoteDao
import com.kkoscevic.noteapp.database.entities.Note

@Database(entities = [Note::class], version = 1,
exportSchema = false)
@TypeConverters(NoteTypeConverters::class)
abstract class NoteDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao
}
```

Prema izvoru [3] anotacija @Dao označava da se radi o klasi kao objektu pristupa podacima. Klasa s @Dao anotacijom je sučelje s funkcijama za svaku operaciju baze podataka koja se želi izvesti. U programskom kodu 4.3 nalaze se funkcije korištene za izradu projekta. @Query anotacija označava da je funkcija namijenjena izvlačenju informacija iz baze. Funkcije za kreiranje, ažuriranje i brisanje podataka imaju anotaciju u skladu sa željenim zadacima. Nakon @Query nalazi se SQL naredba kojom dohvaćamo željene podatke.

Programski kod 4.3: Klasa NoteDao

```
package com.kkoscevic.noteapp.database.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import com.kkoscevic.noteapp.database.entities.Note
import java.util.*

@Dao
interface NoteDao {
    @Query("SELECT * FROM note ORDER BY dateTime DESC")
    fun getNotes(): LiveData<List<Note>>

    @Query("SELECT * FROM note WHERE id=(:id)")
    fun getNote(id: UUID): LiveData<Note?>

    @Query("SELECT * FROM note WHERE title LIKE :searchQuery OR
noteText LIKE :searchQuery")
    fun searchNote(searchQuery: String): LiveData<List<Note>>

    @Insert
    fun addNote(note: Note)

    @Update
    fun updateNote(note: Note)

    @Delete
    fun deleteNote(note: Note)

    @Delete
    fun deleteAllNotes(list: List<Note>)
}
```

Zadnja potrebna klasa za Room bazu podataka je klasa za pretvorbu tipova. Prema izvoru [3] Room koristi SQLite bazu podataka. SQLite je relacijska baza podataka otvorenog koda. Za korištenje Room biblioteke ne trebate znati koristiti SQLite. Pretvornik tipova se koristi iz razloga što neki objekti koji se koriste u Room biblioteci ne mogu se zapisati u SQLite bazu podataka. Za pretvorbu su potrebne dvije funkcije s anotacijom @TypeConverter, vidljivo na 4.4 slici. Jednom funkcijom se pretvara iz Room biblioteke u tip za SQLite. Drugom funkcijom se pretvara iz prikaza u SQLite bazi podataka u izvornu tip.

Programski kod 4.4: Klasa TypeConverter

```
package com.kkoscevic.noteapp.database

import androidx.room.TypeConverter
import java.util.*

class NoteTypeConverters {
    @TypeConverter
    fun toUUID(uuid: String?): UUID? {
        return UUID.fromString(uuid)
    }

    @TypeConverter
    fun fromUUID(uuid: UUID?): String? {
        return uuid?.toString()
    }
}
```

4.2 Spajanje baze podataka s aplikacijom

Za pristup bazi podataka s aplikacije koristi se repozitorij oblikovni obrazac (engl. *repository pattern*). Prema izvoru [3] klasa repozitorija sadrži logiku pristupa podacima iz izvora. Određuje kako dohvatiti i pohraniti određeni skup podataka. Klasa repozitorija je *singleton*. To znači da će u procesu aplikacije postojati samo jedna njena instanca. Da bi klasa bila *singleton* u popratnom objektu mora imati dvije funkcije. Jedna funkcija inicijalizira novu instancu, a druga pristupa repozitoriju, pogledati programski kod 4.5.

Programski kod 4.5: Implementacija klase NoteRepository

```
companion object {
    private var INSTANCE: NoteRepository? = null

    fun initialize(context: Context) {
        if (INSTANCE == null) {
            INSTANCE = NoteRepository(context)
        }
    }

    fun get(): NoteRepository {
        return INSTANCE ?:
        throw IllegalStateException("NoteRepository must be
initialized!")
    }
}
```

Za inicijalizaciju klase *NoteRepository*, čim je aplikacija spremna treba stvoriti klasu namjere aplikacije (engl. *intent application*), pogledati programski kod 4.6. Klasa *NoteIntentApplication* proširuje se s klasom Aplikacija (engl. *application*) i nadjačava funkciju Activity.onCreate() koja je do sada inicijalizirala repozitorij. Prema izvoru [3]

klasa Aplikacija se koristi umjesto klase Aktivnost iz razloga što se instanca aplikacije ne uništava niti stvara poput instance aktivnosti. Instanca aplikacije se stvara pri pokretanju aplikacije i uništava kada se uništi proces aplikacije.

Programski kod 4.6: Klasa NoteIntentApplication

```
package com.kkoscevic.noteapp

import android.app.Application

class NoteIntentApplication :
Application() {
    override fun onCreate() {
        super.onCreate()
        NoteRepository.initialize(this)
    }
}
```

U klasi *NoteRepository* potrebno je napraviti konkretnu implementaciju baze. Za implementaciju baze se koristi funkcija Room.databaseBuilder, pogledati programski kod 4.7. Prema izvoru [9] za funkciju su potrebna tri parametra. Prvi parametar je kontekst. Prema izvoru [10] kontekst je sučelje koje omogućuje pristup resursima i klasama specifičnima za aplikaciju. Drugi parametar je klasa baze podataka koja se želi kreirati. Zadnji parametar je naziv baze podataka koja će se kreirati. Klasa se implementira kao objekt pristupa podacima u klasi *NoteRepository* kako bi se pristupilo operacijama nad bazom podataka. Prema izvoru [3] za bolju sigurnost izvođenja operacija baze podataka koristi se izvršitelj (engl. *executor*). Izvršitelj prihvaća blok koda za izvođenje te izvodi samo taj kod.

Programski kod 4.7: Implementacija baze podataka u klasi NoteRepository

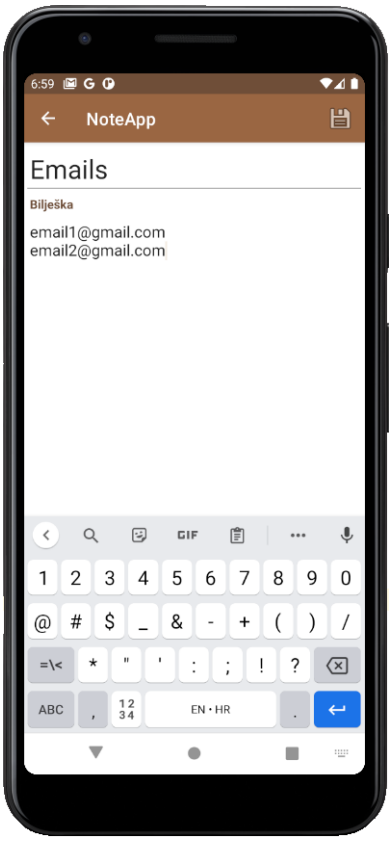
```
private val database: NoteDatabase =
    Room.databaseBuilder(
        context.applicationContext,
        NoteDatabase::class.java,
        DATABASE_NAME
    ).build()

private val executor =
Executors.newSingleThreadExecutor()
private val noteDao = database.noteDao()
```

4.3 Upravljanje bilješkama

Aktivnost je jedna, usredotočena stvar koju korisnik može učiniti. Klasa Aktivnost se brine za stvaranje prikaza korisničkog sučelja. Gotovo sve aktivnosti stupaju u interakciju s

korisnikom. Prema izvoru [11] gotovo sve aktivnosti implementiraju metode *onCreate*, gdje se inicijalizira aktivnost i *onPause*, gdje se definira što se dešava kada korisnik pauzira aplikaciju. Za svaku akciju se koristi zasebna aktivnost. Kada je potrebno koristiti funkcije klase repozitorija tada se koristi model prikaza.

Za kreiranje bilješke korisnik unosi naziv i tekst bilješke. Za spremanje bilješke sa svim podacima koristi se 4.8 programski kod. Ukoliko naziv bilješke nije unesen, naziv bilješke će biti zadan „Bez naziva“ ili „No title“, ovisno o lokalizaciji. Ukoliko se ne unese tekst bilješke, dolazi obavijest i bilješka se ne sprema. Na slici 4.2 je prikazano kako izgleda korisničko sučelje aktivnosti kada je unesen tekst.

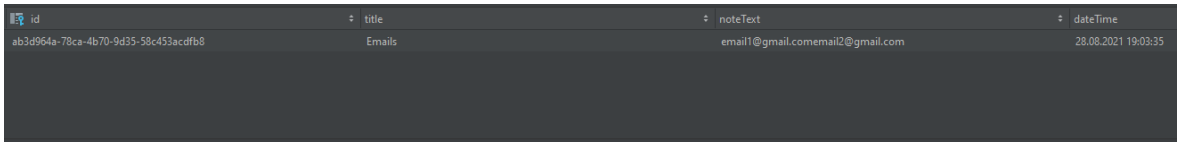


Slika 4.2 Korisničko sučelje za unos bilješke nakon što su uneseni podaci

Programski kod 4.8: Spremanje bilješke u bazu podataka

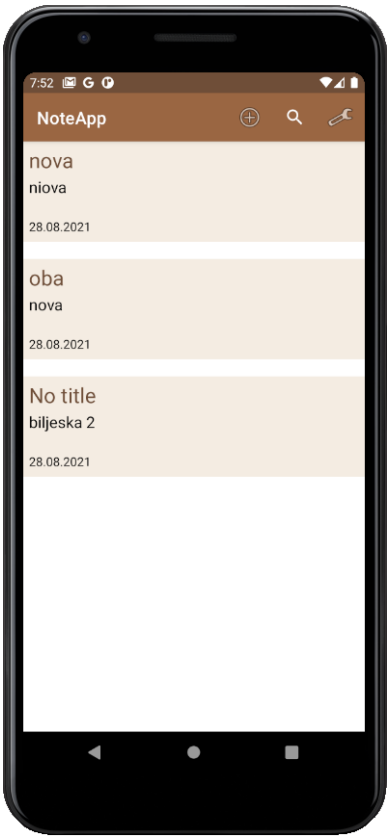
```
note.title = txtNoteTitle.text.toString()
note.noteText = mtxtNoteText.text.toString()
note.dateTime =
LocalDateTime.now().format(dateFormat).toString()
noteViewModel.createNote(note)
Toast.makeText(
    this,
    resources.getString(R.string.note_text_save),
    Toast.LENGTH_SHORT
)
```

Bilješka se sprema klikom na disketu na izborniku. U bazu podataka osim naziva i teksta bilješke sprema se datum kada je bilješka kreirana i njen ID, slika 4.3.



Slika 4.3 Prikaz podataka iz baze podataka

Za prikaz svih bilješki korišten je *RecylerView*. Prema izvoru [3] *RecylerView* prikazuje popis objekata koji se nazivaju prikazima stavki (engl. *item views*). Svaki prikaz stavki predstavlja jedan objekt s popisa podataka. U aplikaciji prikaz stavki sadrži naziv bilješke, dio teksta bilješke i kada je nastala ili zadnje uređivana. Kao što se vidi na slici 4.4 postoji izbornik gdje se nalazi gumb za dodavanje nove bilješke, pretraživanje bilješke te otvaranje postavki. Pretraživanje se odnosi i na naziv i na tekst bilješke, vidljivo na slici 4.5.



Slika 4.4 Korisničko sučelje za prikaz svih bilješki

Slika 4.5 Pretraživanje bilješki prema slovu „b“

U programskom kodu 4.9 vidljiv je način prikazivanja bilješke. U prikazu bilješke ne piše podatak o vremenu kada je nastala bilješka već samo datum. Vrijeme kada je nastala bilješka korišteno je za raspored prikaza. Novije bilješke, kao i one koje su zadnje uređivane nalaze se pri vrhu.

Programski kod 4.9: Prikazivanje podataka svih bilješki

```
this.note = note
tvNoteTitle.text = note.title
tvNoteText.text = note.noteText
val date = LocalDateTime.parse(
    note.dateTime,
    DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")
)
tvDate.text =
date.format(DateTimeFormatter.ofPattern("dd.MM.yyyy"))
```

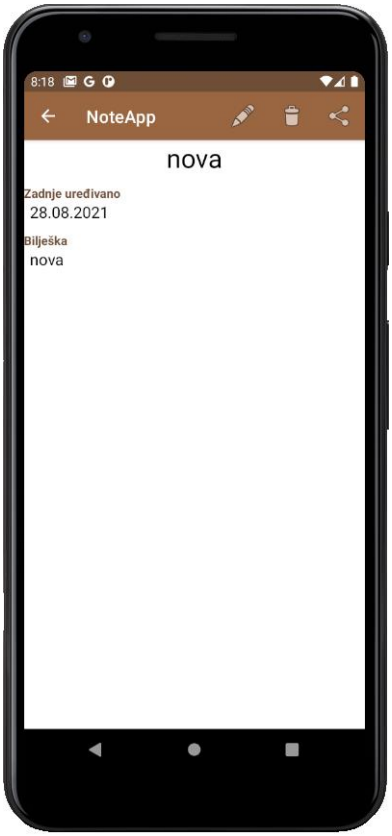
MainActivity, *CreateNoteActivity* i *SettingsActivity* klase koriste NoteV*iewModel*, vidljivo na 4.10 programskom kodu. *MainActivity* klasa koristi ga za dohvaćanje svih bilješki, spremanje bilješki preko Google Asistenta i za pretraživanje bilješki.

CreateNoteActivity klasa ga koristi za spremanje bilješki te *SettingsActivity* klasa za dohvaćanje bilješki kako bi se mogle sve bilješke obrisati.

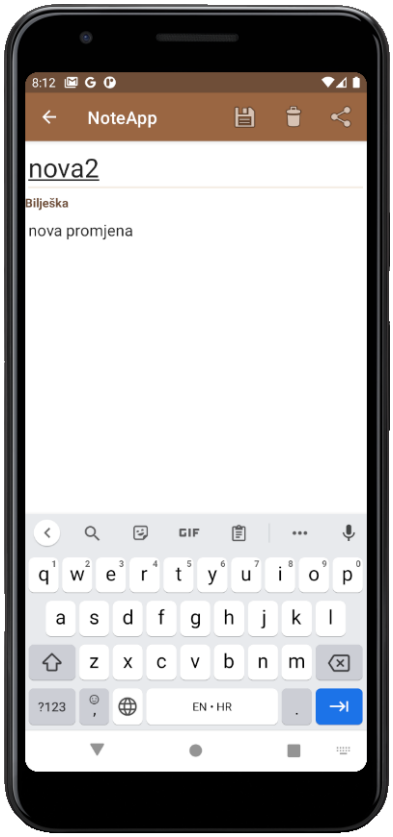
Programski kod 4.10: Klasa NoteViewModel

```
class NoteViewModel : ViewModel() {
    private val noteRepository = NoteRepository.get()

    val notesLiveData = noteRepository.getNotes()

    fun deleteAllNotes(list : List<Note>) {
        noteRepository.deleteAllNotes(list)
    }


    fun searchDatabase(searchQuery: String): LiveData<List<Note>> {
        return noteRepository.searchNote(searchQuery)
    }

    fun createNote(note: Note) {
        noteRepository.addNote(note)
    }

    fun updateNote(note: Note) {
        noteRepository.updateNote(note)
    }
}
```

Za detaljan pregled bilješke potrebno je pritisnuti na bilješku. Kao što se vidi na slici 4.6 nije moguće odmah uređivati bilješku. Postoji izbornik gdje se nalaze gumbi za uredi, obriši i podijeli bilješku. Klikom na uređivanje otvara se korisničko sučelje sličan kreiranju bilješke, slika 4.7. Razlika je što tekst postoji i što se izbornik sada sastoji od gumba spremi, obriši i podijeli bilješku, vidjeti sliku 4.7.



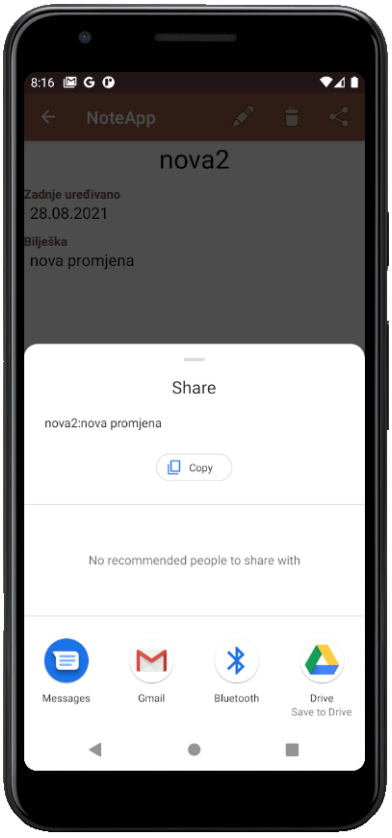
Slika 4.6 Detaljan prikaz bilješke



Slika 4.7 Korisničko sučelje za uređivanje bilješke

Kao što se vidi na slici 4.8. prilikom pritiska na gumb za brisanje na korisničkom sučelju detalji bilješke i uređivanje bilješke otvara se skočni prozor koji provjerava Vašu odluku. Pritiskom na gumb za dijeljenje, slika 4.9, otvara se korisničko sučelje s aplikacijama preko kojih se može podijeliti bilješka. Bilješka se dijeli u obliku „Naziv bilješke: tekst bilješke“. Klase *ShowNoteActivity* i *EditNoteActivity* koriste ShowNoteV*iewModel*, pogledati 4.11 programski kod.

Slika 4.8 Brisanje bilješke



Slika 4.9 Dijeljenje bilješke

Programski kod 4.11: Klasa ShowNoteViewModel

```
class ShowNoteViewModel : ViewModel() {
    private val noteRepository = NoteRepository.get()
    private val noteIdLiveData = MutableLiveData<UUID>()

    var noteLiveData: LiveData<Note?> =
        Transformations.switchMap(noteIdLiveData) { noteId ->
            noteRepository.getNote(noteId)
        }

    fun loadNote(noteId: UUID) {
        noteIdLiveData.value = noteId
    }

    fun saveNote(note: Note) {
        noteRepository.updateNote(note)
    }

    fun deleteNote(note: Note) {
        noteRepository.deleteNote(note)
    }
}
```

4.4 Google virtualni asistent

Za povezivanje aplikacije s Google virtualnim asistentom potrebno je imati korisnički račun na Google Play Konzoli, (engl. *Google Play Console)*. Za korisnički račun na Google Play Konzoli prilikom kreiranja treba platiti 25 američkih dolara. Osim korisničkog računa, potrebno je instalirati *Google Assistant* dodatak kao i alat za ispitivanje radnji aplikacije (engl. *App Actions Test Tool*) dodatak. Alat za ispitivanje radnji akcije se instalira iz razloga što se u projektu koristi actions.xml resursna datoteka. Također, treba napraviti i korisnički račun na *Firebase* platformi jer kasnije će biti potrebne dinamičke veze. Dinamičke veze se koriste za pokretanje logike specifične aplikaciji određene pozivom Google virtualnog asistenta. Nakon prethodnih koraka, aplikacija se treba učitati na Google Play Konzolu. Na mobitelu gdje se testira aplikacija treba biti prijavljen s Google korisničkim računom koji se koristi za Google Play Konzolu. Moguće je izravno s Google virtualnog asistenta pristupiti naredbama aplikacije, a moguće je i s alatom za ispitivanje radnji aplikacije.

Prema izvoru [12] Google virtualni asistent trenutno podržava dvadeset jezika od kojih nijedan nije hrvatski jezik. Sva komunikacija u aplikaciji s Google virtualnim asistentom iz tog razloga se vrši na engleskom jeziku. Također, svaka ugrađena namjera (engl. *built-in intents*, skraćeno BII) koji se koristi ne može se koristiti za svaki jezik. Koje jezike BII podržava piše u BII dokumentu.

Za razgovor s Google virtualnim asistentom koriste se **pisane i glasovne** naredbe. Dalje u tekstu bit će korištene pisane naredbe radi prikaza. Glasovne naredbe nekada nisu posve točne odnosno NLP ne prepozna točno riječ. Do toga dolazi radi naglaska i šumova.

4.4.1 **Otvaranje aplikacije**

Za otvaranje korisničkog sučelja aplikacije koristi se Google ugrađena namjera naziva „Otvorite značajku aplikacije“, (engl. *open app feature*). Prema izvoru [13] naredbe kojima se može otvoriti korisničko sučelje su sljedeće:

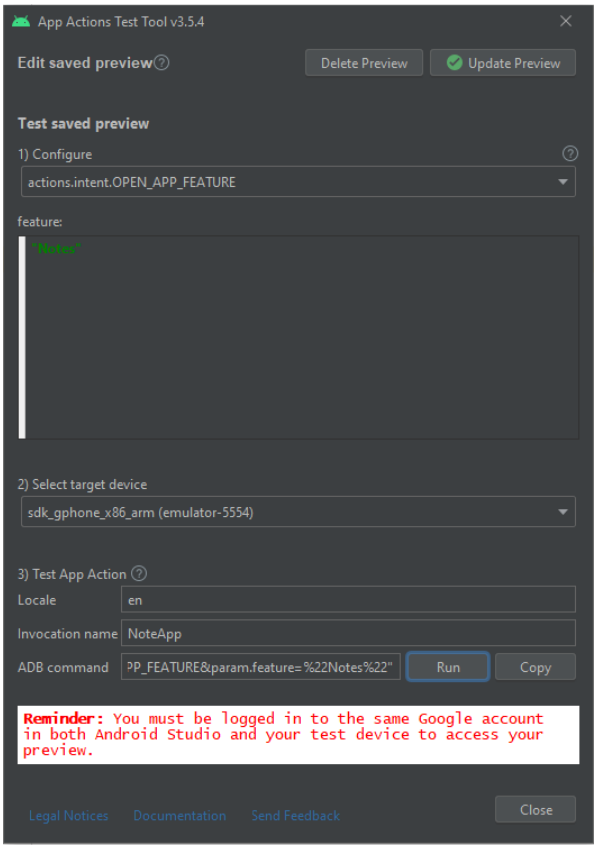
- Open notes in NoteApp.
- Show notes on NoteApp.

U programskom kodu 4.12 prikazan je način korištenja *OPEN_APP_FEATURE* u izgradnji aplikacije. Unutar tagova <*entity-set*> se nalaze svi nazivi korisničkih sučelja koje je moguće otvoriti. „urlTemplate“ je dinamička veza stvorena na stranici *Firebase*. Za svaku namjeru se koristi nova dinamička veza.

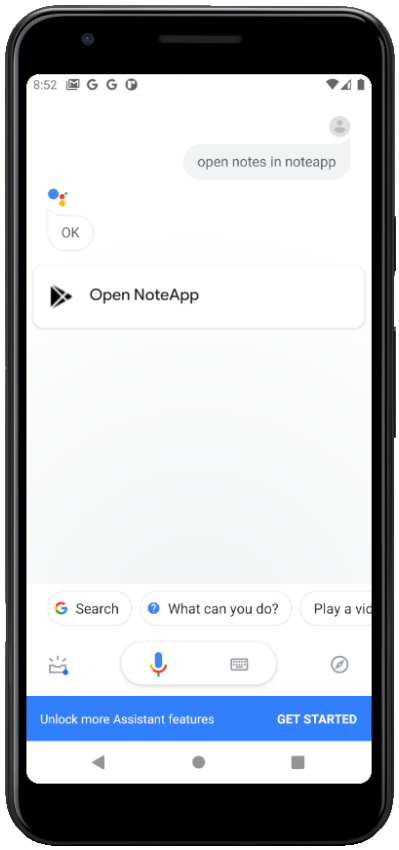
Programski kod 4.12: Ugrađena namjera OPEN_APP_FEATURE

```
<action intentName="actions.intent.OPEN_APP_FEATURE">
    <fulfillment
urlTemplate="https://kkoscevic.page.link/open{?featureName}">
        <parameter-mapping
            intentParameter="feature"
            urlParameter="featureName" />
    </fulfillment>
    <parameter name="feature">
        <entity-set-reference entitySetId="featureEntitySet" />
    </parameter>
</action>

<entity-set entitySetId="featureEntitySet">
    <entity
        name="Notes"
        identifier="notes" />
    <entity
        name="Settings"
        identifier="settings" />
</entity-set>
```

Na slici 4.10 prikazan je alat za ispitivanje radnji aplikacije. Alat za ispitivanje radnji aplikacije omogućuje pristup Google virtualnom asistentu. Prilikom pokretanja alata za ispitivanje radnji aplikacije unosi se koju lokalizaciju želimo, zadano je engleski jezik. Zatim se odabere naziv poziva (engl. *invocation name*), zadano je „*test app action*“. Naziv poziva je moguće staviti bilo što, dalje u tekstu će biti *NoteApp*. Za otvaranje korisničkog sučelja potrebno je unijeti samo jedan parametar u alat za ispitivanje radnji aplikacije. Na slici 4.11 se nalazi naredba kojom se otvara korisničko sučelje.



Slika 4.10 Otvaranje korisničkog sučelja preko Google virtualni asistenta pomoću alata za ispitivanje radnji aplikacije



Slika 4.11 Otvaranje korisničkog sučelja preko Google virtualnog asistenta

4.4.2 **Pretraživanje aplikacije**

Za pretraživanje bilješki koristi se Google ugrađena namjera naziva „Dohvati stvar“, (engl. *Get thing*). Prema izvoru [14] naredbe kojima se može pretražiti su sljedeće:

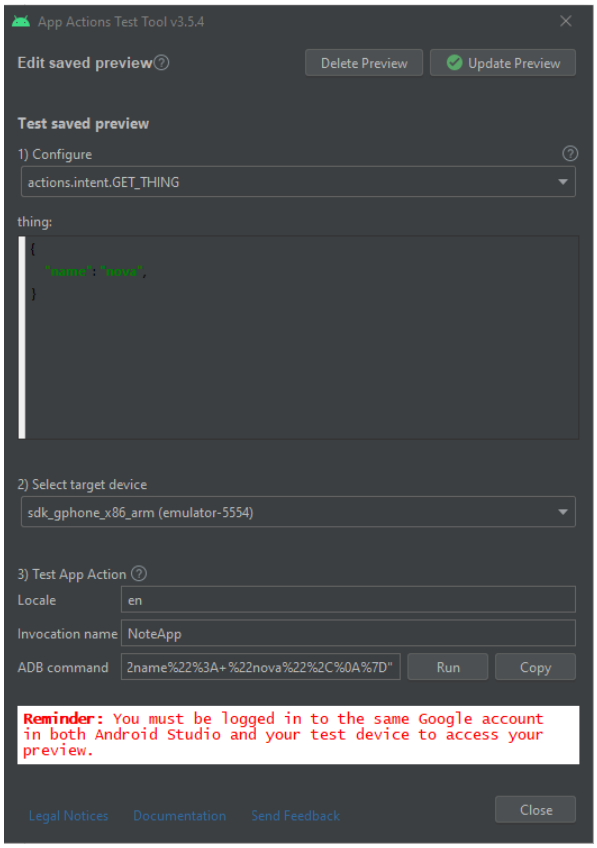
- Search for b in NoteApp.
- Look up no title in NoteApp..
- Find no in NoteApp.

U programskom kodu 4.13 prikazan je način korištenja *GET_THING* u izgradnji aplikacije. Za razliku od *OPEN_APP_FEATURE* jedini parametar je *q*. Parametar *q* je unos koji želite pretražiti.

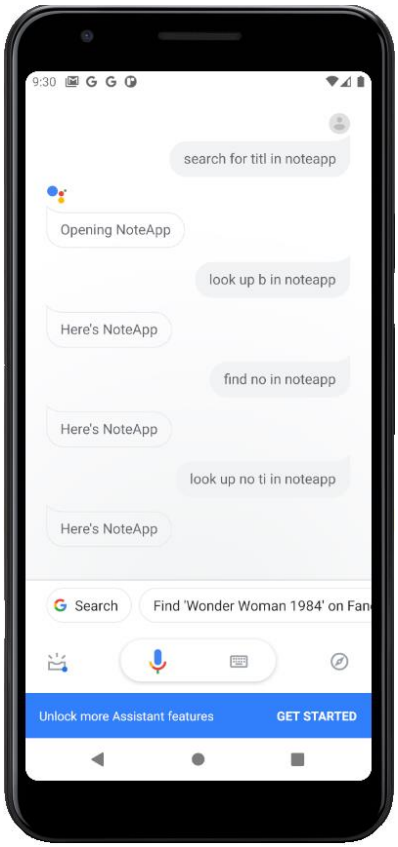
Programski kod 4.13: Ugrađena namjera GET_THING

```
<action intentName="actions.intent.GET_THING">
    <fulfillment
urlTemplate="https://kkoscevic.page.link/search{?q}">
        <parameter-mapping
            intentParameter="thing.name"
            urlParameter="q" />
    </fulfillment>
</action>
```

Na slici 4.12 je potreban jedan parametar koji je označen kao *name*. To označava da se radi o nazivu stvari. Bilješke se pretražuju prema tom parametru. Na slici 4.13 su prikazane naredbe koje se koriste za pretraživanje.



Slika 4.12 Pretraživanje bilješke preko Google virtualnog asistenta pomoću alata za ispitivanje radnji aplikacije



Slika 4.13 Pretraživanje bilješke preko Google virtualnog asistenta

4.4.3 **Spremanje bilješke**

Za spremanje bilješki koristi se Google ugrađena namjera naziva „Stvori stvar“, (engl. *Create thing*). Prema izvoru [15] naredbe kojima se može spremiti bilješka su sljedeće:

- Save cat to NoteApp

- Add monkey to NoteApp

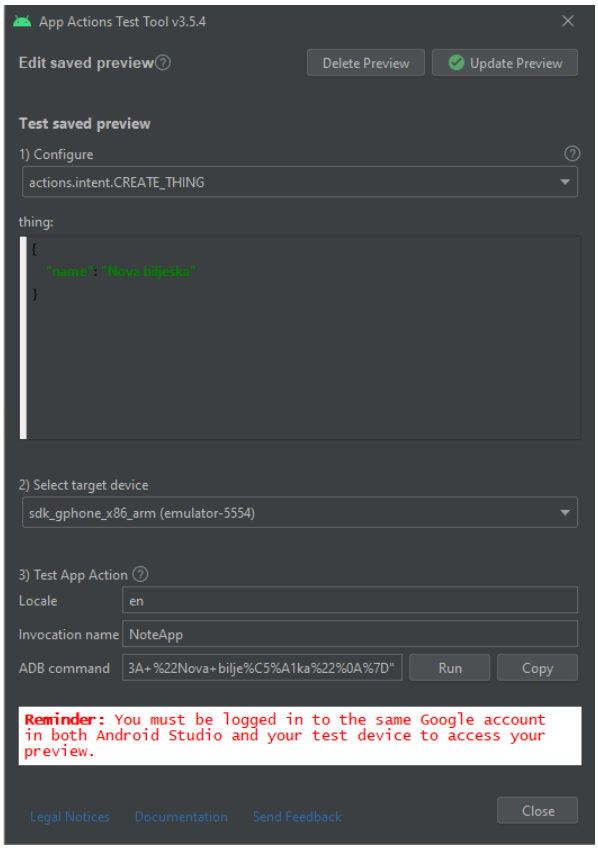
U programskom kodu 4.14 prikazan je način korištenja *CREATE_THING* u izgradnji aplikacije. Sadrži parametar *name* koji je u aplikaciji zapravo tekst bilješke. Ukoliko se ne unese parametar, Google virtualni asistent ne bude unio bilješku već samo otvorio korisničko sučelje sa svim bilješkama. Ukoliko se kao parametar unese riječ koja nije na engleskom Google virtualni asistent je prevodi na engleski i sprema u bilješku.

Programski kod 4.14: Ugrađena namjera CREATE_THING

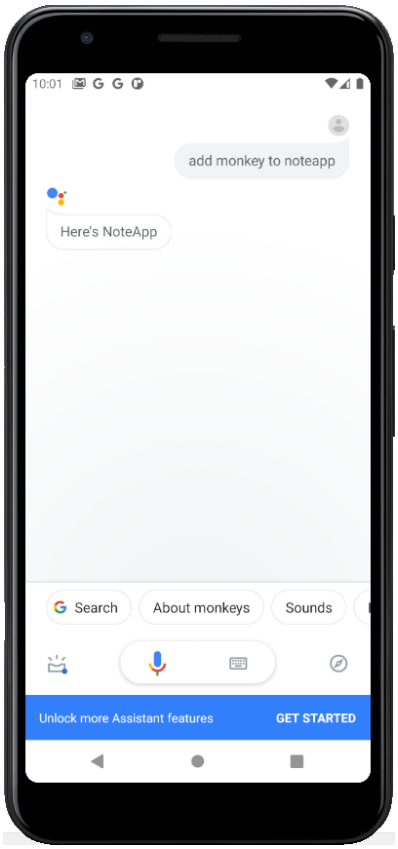
```
<action intentName="actions.intent.CREATE_THING">
    <fulfillment
urlTemplate="https://kkoscevic.page.link/create{?name}">
        <parameter-mapping
            intentParameter="thing.name"
            required="true"
            urlParameter="name" />
    </fulfillment>
    <fulfillment
urlTemplate="https://kkoscevic.page.link/notes" />
</action>
```

Prilikom spremanja bilješke preko Google virtualnog asistenta ukoliko riječ nema smisla on ne stvara bilješku ili stvara bilješku sa pojmom najbližim unesenom parametru. Također, problem je i što ne želi spremiti bilješku ukoliko se unese više od jedne riječi. Još jedan problem je što ne prepoznaje dodatni parametar ukoliko se postavi. Do problema dolazi jer NLP još nije najbolje razvijen te mogućih problema u kodu ugrađenih namjera. Ukoliko se koristi alat za ispitivanje radnji aplikacije do tih problema ne dolazi.

Na slici 4.14 je potreban jedan parametar koji je označen kao *name*. To označava da se radi o nazivu stvari. Vrijednost parametra *name* se unosi kao tekst bilješke, a naziv bilješke je „Bez naziva“ ili „No title“ ovisno o lokalizaciji. Na slici 4.15 je prikazana naredba za spremanje bilješke preko Google virtualnog asistenta.



Slika 4.14 Spremanje bilješke preko Google virtualnog asistenta pomoću alata za ispitivanje radnji aplikacije



Slika 4.15 Spremanje bilješke preko Google virtualnog asistenta

5. ZAKLJUČAK

Tema završnog rada je stvaranje mobilne aplikacije za Android operativni sustav kojoj je cilj interakcija s Google virtualnim asistentom. Google virtualni asistent nije toliko poznat zbog čega nije mnogo korišten. Funkcije Google virtualnog asistenta korištene u interakciji s aplikacijom su otvaranje korisničkog sučelja, pretraživanje i spremanje bilješki. Kompleksnost implementacije Google asistenta ovisi jesu li potrebne postojeće namjere ili treba nešto novo. Problemi Google virtualnog asistenta se pojavljuju pri obradi prirodnog jezika. Prirodni jezik, jezik koji ljudi koriste pri svakodnevnoj komunikaciji s drugima, se pojavljuje u uvjetima koji su i manje nego idealni za kvalitetan rad aplikacija kao što su Google virtualni asistent. U procesu razmjene prirodnog jezika može doći do raznih problema - naglasci individualaca, pozadinska buka, postojanost idioma i dvosmislenih rečenica samo su neki od problema zbog kojih je potreban njegov daljnji razvoj. Tijekom posljednjeg desetljeća došlo je do velikog napretka u obradi prirodnog jezika, ali je još mnogo istraživanja potrebno kako bi se postigao besprijekoran rad aplikacija.

Tijekom procesa rada na aplikaciji stvorenoj za završni rad pojavilo se nekoliko problema. Jezik ne ostaje isti pri promijeni zadanog jezika u postavkama nakon ponovnog pokretanja aplikacije. Ostali problemi vezani su uz Google virtualni asistent. Postoji relativno malo dokumentacije o samoj implementaciji Google virtualnog asistenta kao i o ugrađenim namjerama zbog čega dolazi do zastoja u rješavanju potencijalnih problema. Upravo taj manjak dokumentacije rad s Google virtualnim asistentom čini i zanimljiv. Kako bi se problem riješio treba se uložiti više truda i samostalnog istraživanja koda. Kod Google virtualnog asistenta problem je predstavljala dinamička veza čiji sam izgled nije bitan – bitna je funkcionalnost i korist. Sljedeći problem Google virtualnog asistenta vezan je uz stvaranje bilješki. Nakon dužeg perioda proučavanja, zaključak je da problem nastaje ako je riječ nerazumna. Da bi bilješka bila unesena, uneseni parametar mora imati smisla Google virtualnom asistentu. Spremanje više od jedne riječi preko Google virtualnog asistenta dovodi do novog problema. Problem je uzrokovan manjkom razvoja NLP i koda koji se nalazi iza ugrađenih namjera. Problem nažalost nije riješen.

Iako postoje navedeni problemi, stvorena aplikacija je zbog korištenja Google virtualnog asistenta učitana na Google Play Konzolu. Rezultat toga je da se aplikacija može postaviti na Google Trgovinu na kojoj može postati dostupna široj javnosti.

Odgovori na pitanja o sadržaju aplikacije, određeni testeri, postavljena politika privatnosti, te određena regija gdje je aplikacija dostupna samo su neki od uvjeta koje je potrebno zadovoljiti kako bi Google Play objavio aplikaciju.

6. LITERATURA

[1] J. Chen, »Investopedia,« 3 Veljača 2021. [Mrežno]. Available: https://www.investopedia.com/terms/a/android-operating-system.asp. [Pokušaj pristupa 10 Kolovoz 2021].

[2] »El-pro-cus,« 2020. [Mrežno]. Available: https://www.elprocus.com/what-is-android-introduction-features-applications/. [Pokušaj pristupa 10 Kolovoz 2021].

[3] B. G. B. P. C. S. K. Marsicano, Android Programming: The Big Nerd Ranch Guide, 4. ur., Big Nerd Ranch, LLC, 2019.

[4] »Kotlin,« 11 Veljača 2021. [Mrežno]. Available: https://kotlinlang.org/docs/android-overview.html. [Pokušaj pristupa 10 Kolovoz 2021].

[5] »Android Developers,« [Mrežno]. Available: https://developer.android.com/studio/intro. [Pokušaj pristupa 10 Kolovoz 2021].

[6] M. Tillman, »Pocket-lint,« 25 Ožujak 2021. [Mrežno]. Available: https://www.pocket-lint.com/apps/news/google/137722-what-is-google-assistant-how-does-it-work-and-which-devices-offer-it. [Pokušaj pristupa 10 Kolovoz 2021].

[7] M. B. G. A. S. H. Vajjala S., u *Practical Natural Langugae Processing: A Comprehensive Guide to Building Real-World NLP Systems*, O'Reilley Media, Inc, 2020, pp. 3-33.

[8] »Android Developers,« [Mrežno]. Available: https://developer.android.com/jetpack/guide#connect-viewmodel-repository. [Pokušaj pristupa 29 Kolovoz 2021].

[9] »Android Developers,« [Mrežno]. Available: https://developer.android.com/reference/android/arch/persistence/room/Room. [Pokušaj pristupa 28 Kolovoz 2021].

[10] »Android Developers,« [Mrežno]. Available: https://developer.android.com/reference/android/content/Context. [Pokušaj pristupa 28 kolovoz 2021].

[11] »Android Developers,« [Mrežno]. Available: https://developer.android.com/reference/android/app/Activity. [Pokušaj pristupa 28 Kolovoz 2021].

[12] »Google Assistant,« [Mrežno]. Available: https://developers.google.com/assistant/console/languages-locales. [Pokušaj pristupa 28 Kolovoz 2021].

[13] »Google Assistant,« [Mrežno]. Available: https://developers.google.com/assistant/app/reference/built-in-intents/common/open-app-feature?hl=en. [Pokušaj pristupa 29 Kolovoz 2021].

[14] »Google Assistant,« [Mrežno]. Available: https://developers.google.com/assistant/app/reference/built-in-intents/common/get-thing?hl=en. [Pokušaj pristupa 29 Kolovoz 2021].

[15] »Google Assistant,« [Mrežno]. Available: https://developers.google.com/assistant/app/reference/built-in-intents/common/create-thing?hl=en. [Pokušaj pristupa 29 Kolovoz 2021].

[16] P. Saccomani, »MobiLoud,« [Mrežno]. Available: https://www.mobiloud.com/blog/native-web-or-hybrid-apps#:~:text=A%20native%20app%2C%20or%20native,apps%20are%20written%20i

n%20Java.. [Pokušaj pristupa 10 Kolovoz 2021].

[17] A. Hayes, »Investopedia,« 14 Lipanj 2020. [Mrežno]. Available: https://www.investopedia.com/terms/b/bayes-theorem.asp. [Pokušaj pristupa 12. kolovoz 2021].

[18] »Android,« [Mrežno]. Available: https://www.android.com/everyone/facts/. [Pokušaj pristupa 10 Kolovoz 2021].

[19] »Android Developers,« [Mrežno]. Available: https://developer.android.com/reference/java/lang/String. [Pokušaj pristupa 28 Kolovoz 2021].

[20] »Google Assistant,« [Mrežno]. Available: https://developers.google.com/assistant/app/reference/built-in-intents/common/open-app-feature?hl=en#actionsxml. [Pokušaj pristupa 29 Kolovoz 2021].

7. OZNAKE I KRATICE

VM – virtualni stroj (engl. *Virtual machine*)

NLP – obrada prirodnog jezika (engl. *Natural language processing*)

Regex – regularni izrazi

CFG – gramatika bez konteksta (engl. *Context-free grammar*)

SVM – stroj s potpornim vektorima (engl. *Support vector machines*)

HMM – skriveni Markovljev model (engl. *Hidden Markov model*)

CRF – uvjetno slučajno polje (engl. *Conditional random field*)

RNN – ponavljajuće neuronske mreže (engl. *Recurrent neural network*)

LSTM – dugotrajno kratkotrajno pamćenje (engl. *Long short-term memory*)

CNN – konvolucijske neuronske mreže (engl. *Convolutional neural network*)

API – sučelje za programiranje aplikacija (engl. *Application programming interface*)

ID – jedinstveni identifikatora

UUID – univerzalni jedinstveni identifikator (engl. *universally unique identifier*)

Dao – objekt za pristup podacima (engl. *data access object*)

8. SAŽETAK

Mobilna aplikacija za interakciju s Google virtualnim asistentom

U ovom radu opisana je izrada mobilne aplikacije za prikazivanje, stvaranje i uređivanje postojećih bilješki. Aplikacija, koja je izgrađena u razvojnom okruženju Android Studio koristeći Kotlin programski jezik i za čiju je bazu podataka korištena Room biblioteka, ima mogućnost interakcije s Google virtualnim asistentom. Preko Google virtualnog asistenta moguće je otvaranje prikaza svih bilješki kao i pretraživanje istih po unesenim parametrima, te stvaranje novih. Za sve interakcije aplikacije i Google virtualnog asistenta su korištene ugrađene namjere. Google virtualni asistent prepoznaje zadane naredbe pomoću obrade prirodnog jezika. Agent za razgovor je zadatak obrade prirodnog jezika pomoću kojeg Google virtualni asistent razumije naredbu.

Ključne riječi: Android, Google virtualni asistent, obrada prirodnog jezika, Room

9. ABSTRACT

Mobile application for interaction with Google virtual assistant

This paper describes the development of a mobile notes management application. Specifically, the focus is on displaying notes, creating new, and editing previously existing ones. The application, which was built in the Android Studio using the Kotlin programming language and which contains resources found in The Room library for its database, has the ability to interact with Google Virtual Assistant. Through Google Virtual Assistant, it is possible to open the display of all notes alongside settings, as well as search all notes by entered parameters and create new notes. Built-in intents were used for all interaction between applications and Google Virtual Assistant. Google Virtual Assistant recognizes default commands using natural language processing. The conversational agent is a natural language processing task by which Google Virtual Assistant understands a command.

Keywords: Android, Google Virtual Assistant, natural language processing, Room

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 27. kolovoza 2021	Klara Košćević	Klara Košćević

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Klara Koščević

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 17. kolovoz 2021

Klara Koščević

potpis studenta/ice