Aplikacija za prevođenje govora i teksta

Šumiga, Marin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

Aplikacija za prevođenje teksta i govora

Završni rad

Marin Šumiga

Osijek, 2024.

SADRŽAJ

1. UVOD

U današnjem svijetu, sposobnost učinkovite komunikacije na različitim jezicima postaje sve važnija. Mobilne aplikacije za prevođenje jezika igraju ključnu ulogu u prevladavanju jezičnih barijera, omogućujući korisnicima brz i jednostavan pristup prijevodima u raznim situacijama. Razvoj tehnologije, posebice u području umjetne inteligencije i strojnog učenja, značajno je unaprijedio kvalitetu i dostupnost automatiziranih prijevoda [1]. Međutim, unatoč brojnim postojećim rješenjima i dalje postoji prostor za inovacije posebno u pogledu personalizacije korisničkog iskustva.

Automatsko prepoznavanje govora (eng. *Automatic-Speech-Recognition - ASR*), kao i modeli strojnog prevođenja igraju ključnu ulogu u tradicionalnom (eng. *Speech-to-Text - ST*) prevođenju, omogućavajući pretvaranje govornog jezika u izvornom obliku u pisani tekst i olakšavanje međujezične komunikacije. ASR prepoznaje izgovorene riječi, dok model strojnog prevođenja prevodi transkribirani tekst na ciljni jezik [2].

Rad se sastoji od pet poglavlja. Prvo poglavlje predstavlja idejni plan za izradu aplikacije i završnog rada. U drugom se poglavlju uspoređuju trenutno dostupne aplikacije na tržištu koje pokušavaju riješiti isti problem kao aplikacija izrađena za potrebe ovog završnog rada. U trećem su poglavlju navedene tehnologije i alati korišteni pri izradi aplikacije. Četvrto poglavlje opisuje proces izrade aplikacije kao i njene funkcionalnosti. Rad završava s petim poglavljem koji je ujedno i zaključak ovog završnog rada.

1.1. Zadatak završnog rada

U okviru ovog završnog rada izradit će se mobilna aplikacija za operacijski sustav Android koja će korisnicima omogućiti prevođenje teksta, govora i teksta sa slike na različite jezike. Aplikacija će omogućiti prijavu i odjavu korisnika kako bi mogli spremati svoje omiljene prijevode za brzi pristup. Za razvoj aplikacije koristit će se Android Studio, programski jezik Kotlin i Jetpack Compose. Podaci će se pohranjivati u Firebase Firestore bazu podataka, koja će omogućiti spremanje prijevoda i korisničkih postavki te podržavati funkcionalnosti prijave i odjave. Za prevođenje će se koristiti Googleov ML Kit.

2. ISTRAŽIVANJE POSTOJEĆIH RJEŠENJA

U ovom će poglavlju biti opisana neka od postojećih rješenja koji imaju istu svrhu kao i aplikacija izrađena za potrebe ovog završnog rada.

2.1. Google prevoditelj

Google prevoditelj jedan je od najpopularnijih i najkorištenijih alata za prevođenje na svijetu, a temelji se na strojnom učenju. Lansiran 2006. godine, ovaj besplatni servis nudi prevođenje teksta, govora, dokumenata i mrežnih stranica na više od 100 jezika, uključujući i razne rijetke jezike i dijalekte [3].

Strojno prevođenje bliskih jezika jednostavnih gramatika jednostavno je, odnosno prijevodi dobiveni strojnim prevođenjem jezika jednostavnih gramatika često se vrlo malo razlikuju od prijevoda dobivenih ljudskim prevođenjem, npr. prevođenje s danskog na engleski jezik ili u suprotnom smjeru. Alati za strojno prevođenje nesrodnih jezika složenih gramatika početkom 21. stoljeća ne daju zadovoljavajuće rezultate, pri čemu *Google prevoditelj* bitno ne odstupa od drugih takvih proizvoda [4].

Iako ima nedostataka, služi kao korektan alat za prevođenje. Može mu se pristupiti preko Google računa, dakle preko preglednika, ali i putem posebne aplikacije, što su njegove prednosti. Velika mu je prednost što je potpuno besplatan za sve korisnike.

2.2. Microsoft Translator

Microsoft Translator [5] koristi napredne tehnologije strojnog učenja, uključujući i neuronske mreže za poboljšanje kvalitete prijevoda. Implementira tehnologiju "zero-shot" prevođenja koja omogućuje prevođenje između jezičnih parova za koje sustav nije izravno treniran, to jest za jezike s kojima nije bio dosad upoznat.

Microsoft Translator je dio Microsoft Cognitive Services i integriran je u više potrošačkih, razvojnih i poslovnih proizvoda, uključujući Bing, Microsoft Office, SharePoint, Microsoft Edge, itd. [5].

Microsoft Translator nudi prijevod teksta i govora putem usluga u oblaku. Usluga za prevođenje teksta putem Translator Text API-ja kreće se od besplatne razine koja podržava dva milijuna znakova mjesečno do plaćenih razina koje podržavaju milijarde znakova mjesečno.

Prijevod govora putem Microsoftovih govornih usluga nudi se na temelju vremena audio sustava. Namijenjen je malim i srednjim poduzećima s ograničenim budžetom, pojedincima koji trebaju brze i jednostavne prijevode te putnicima koji trebaju brze prijevode u stvarnom vremenu. Besplatna verzija dostupna je svima za osnovnu uporabu putem web sučelja i mobilnih aplikacija, a plaćena verzija postoji za naprednije značajke i veće količine prijevoda.

Prednosti su mu što je integriran s drugim Microsoft proizvodima poput prijevoda unutar Microsoft Worda, Excela i PowerPointa. Također, Microsoft Edge web preglednik ima ugrađenu funkciju prevođenja mrežnih stranica koristeći Microsoft Translate. Nudi različite mogućnosti prijevoda (tekst, govor i slika), a API je dostupan razvojnim programerima za korištenje u vlastitim projektima. Nedostaci su kvaliteta prijevoda za manje zastupljene jezike, ograničen broj znakova za unos i potencijalni problemi privatnosti pri korištenju usluge u oblaku.

2.3. Yandex Translate

Yandex Translate [6] je usluga strojnog prevođenja koju pruža ruska tehnološka tvrtka *Yandex*. Ovaj alat podržava prevođenje od oko 95 jezika i nudi neke jedinstvene značajke, posebno za ruski jezik i druge slavenske jezike. *Yandex Translate* koristi vlastitu tehnologiju strojnog prevođenja temeljenu na statističkim metodama i neuronskim mrežama. Sustav se kontinuirano ažurira iz velikoj broja paralelnih tekstova i korisničkih povratnih informacija. Kako bi preveo tekst, računalo ga prvo uspoređuje s bazom podataka riječi. Računalo zatim uspoređuje tekst s modelima osnovnog jezika, pokušavajući odrediti značenje izraza u kontekstu teksta [6].

Yandex Translate je posebno popularan među putnicima i poslovnim ljudima koji trebaju brze i pouzdane prijevode, osobito kada je riječ o ruskom jeziku. Također je koristan za male tvrtke koje posluju na ruskom tržištu ili s ruskim partnerima.

Jedna od glavnih prednosti ovog alata je njegova točnost pri prevođenju između ruskog i drugih jezika. Osim toga, nudi i prepoznavanje teksta na slikama, što može biti iznimno korisno pri putovanjima ili u situacijama kada se susrećete s tekstom na stranom jeziku u stvarnom svijetu. Međutim, ima i svoje nedostatke. Iako je vrlo dobar i pouzdan za ruski jezik, kvaliteta prijevoda može varirati za druge jezične parove. Također, kao i kod mnogih ostalih besplatnih usluga prevođenja, postoje ograničenja u količini teksta koji se može prevesti.

2.4. Naver Papago – AI Translator

Naver Papago je prevoditeljski alat temeljen na umjetnoj inteligenciji, a razvila ga je južnokorejska tvrtka Naver Corporation. Aplikacija koristi napredne tehnologije strojnog učenja za prevođenje teksta, govora i slika između 13 jezika [7]. Korištenje je alata besplatno, ali postoji i plaćena verzija namijenjena za tvrtke.

Prednosti Naver Papaga uključuju široku podršku za azijske jezike, posebno korejski, japanski i kineski, što ga čini izvrsnim za korisnike koji rade s tim jezicima. Aplikacija također nudi prijevode u stvarnom vremenu i prevođenje teksta sa slika. Međutim, ima i nedostataka, a to je manja preciznost za ne azijske jezike i složenije korisničko sučelje za navigaciju u usporedbi s nekim jednostavnijim prevoditeljima.

2.5. DeepL: translate & write

DeepL: translate & writed [8] je alat za prevođenje i pisanje koji koristi duboko učenje za proizvodnju visokokvalitetnih prijevoda. Razvila ga je njemačka tvrtka DeepL GmbH, a ovaj je alat poznat po tome što prevodi prirodnije i tečnije u usporedbi s konkurentima.

Glavna prednost DeepL-a je njegova točnost, pogotovo za europske jezike. Alat također nudi integraciju s različitim softverskim aplikacijama poput ekstenzija za Google Chrome i Microsoft Edge, što ga čini korisnim za profesionalne prevoditelje. DeepL je nedavno uveo funkciju "Write" koja omogućuje formuliranje i preoblikovanje teksta. Nedostaci uključuju ograničen broj podržanih jezika koji trenutno iznosi oko 30 jezika, posebno kada su u pitanju azijski jezici.
Besplatna verzija ima ograničen broj od 500000 znakova koje može prevesti u mjesec dana, dok plaćena verzija, koja iznosi 4,99 eura mjesečno, pruža neograničen broj znakova i maksimalnu sigurnost podatka [9].

2.6. Osvrt na postojeća rješenja

Google prevoditelj, Microsoft Translator, Yandex Translate, Naver Papago i DeepL predstavljaju vodeće alate za strojno prevođenje od kojih svaki ima drugačije karakteristike i primjene.

Google prevoditelj ističe se širokom dostupnošću i najvećom podrškom jezika od svih navedenih alata za prevođenje. Njegova je prednost u tome što je potpuno besplatan i lako dostupan svima, bilo putem preglednika ili mobilne aplikacije.

Microsoft Translator koristi napredne tehnologije strojnog učenja, uključujući neuronske mreže i "zero-shot" prevođenje. Integracija s drugim Microsoft proizvodima čini ga posebno korisnim za poslovne korisnike.

Svaki od ovih alata ima svoje prednosti ovisno o specifičnim zahtjevima i potrebama korisnika. Yandex Translate je izvrstan za ruski i slavenske jezike, Naver Papago je idealan za azijske jezike, a DeepL je najbolji izbor za visokokvalitetne prijevode europskih jezika.

Aplikacija izrađena za potrebe ovog završnog rada bit će besplatna za korištenje i sadržavat će prijevode teksta, govora te tekstove sa slike u stvarnom vremenu. Bit će potpuno transparentna kako bi se neutralizirala zabrinutost nekih korisnika prema tehnološkim rješenjima.

3. KORIŠTENE TEHNOLOGIJE

3.1. Android

Za potrebe ovog završnog rada izrađena je mobilna aplikacija za operacijski sustav Android [10] jer je on najkorišteniji mobilni operacijski sustav, čime se osigurava dostupnost što većem broju korisnika. Aplikacija je testirana na zadnjim dvjema verzijama (Android 13 Tiramisu, Android 14 Upside Down Cake) Android operacijskog sustava jer se ostale zbog zastarjelosti ne koriste više.

3.2. Android Studio

Android Studio [11] je službeno integrirano razvojno okruženje za Android operacijski sustav, temeljen na JetBrainsovom IntelliJ IDEA softveru dizajniranom posebno za razvoj Android aplikacija. Zbog toga i mogućnosti koje pruža kroz razne alate i aplikacije za brže i učinkovitije pisanje koda odabran je za izradu aplikacije. Osim navedenog, ovo razvojno okruženje pruža mnogo mogućnosti pri uređivanju i pisanju koda, virtualne emulatore, uređaje za testiranje aplikacije te mogućnost povezivanja s vlastitim mobilnim uređajem.

3.3. Kotlin

Programski jezik Kotlin [12] je primarni jezik koji se koristio za razvoj ove aplikacije koji je, za razliku od Jave, puno jednostavniji i sažetiji. Omogućuje postizanje istih funkcionalnosti sa znatno manje koda te, ukoliko je je potrebno, i korištenja Jave.

3.4. Jetpack Compose

Jetpack Compose [13] je moderan način izgradnje korisničkog sučelja koju pojednostavljuje i ubrzava pomoću svojih moćnih alata. Mnogo je intuitivniji od prethodnog Extensible Markup Language (XML) koji se koristio za korisnička sučelja u ranijim aplikacijama pisanim u Kotlinu. Jetpack Compose je korišten za izradu korisničkog sučelja predmetne aplikacije, čime se izradilo korisničko sučelje koje je lagano za uporabu novom korisniku aplikacije koja je modernog i minimalističkog izgleda.

3.5. Firebase

Firebase [14] je korišten za pružanje pozadinskih usluga unutar aplikacije od kojih su korištene Cloud Firestore i Firebase Authentication.

3.5.1. Cloud Firestore

Cloud Firestore [15] je NoSQL baza podataka u stvarnom vremenu koja je korištena za pohranu i sinkronizaciju podataka o spremljenim prijevodima i korisničkim računima. Njegova struktura, koja se temelji na dokumentima i kolekcijama, omogućila je fleksibilno modeliranje i upravljanje podacima. Prijevodi koje korisnik odabere spremaju se u bazu podataka odakle se kasnije dohvaćaju i čitko izlistavaju na ekranu. Također, korisnički se podaci spremaju u istu bazu podataka te su povezani s prijevodima, što znači da ukoliko se obriše korisnički račun, izbrisat će se i svi spremljeni prijevodi povezani s tim računom.

3.5.2. Firebase Authentication

Firebase Authentication [16] je korišten za autentifikaciju korisnika unutar ove aplikacije. Ova usluga omogućava jednostavnu implementaciju različitih metoda prijave korisnika unutar aplikacije. Metoda koja je korištena za prijavu korisnika u ovom radu je prijava e-poštom i lozinkom iako Firebase Authentication pruža mogućnosti prijave kao što su Google, Facebook i drugo.

3.6. Google ML Kit

Google ML Kit [17] je softverski razvojni komplet koji pruža pristup unaprijed obučenim modelima strojnog učenja koje je razvio Google. Unutar ove aplikacije korišten je model strojnog učenja namijenjen prevođenju teksta koji je unio korisnik. Također, korišteni su modeli za prepoznavanje teksta sa slika te utvrđivanje jezika određenog teksta.

3.7. CameraX

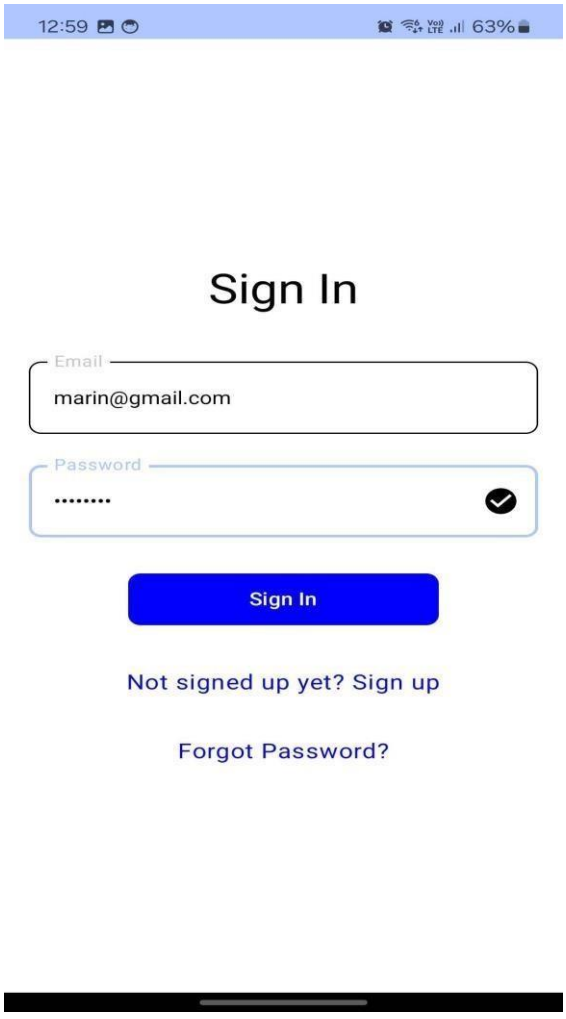
CameraX [18] je Jetpack biblioteka, napravljena da olakša razvoj aplikacija za kameru. Ima podršku za Android 5.0 pa sve do najnovije verzije Android 14.0. CameraX naglašava slučajeve upotrebe koji pomažu u zadatku koji treba obaviti umjesto da se upravlja specifikacijama specifičnim za uređaj.

4. IZRADA APLIKACIJE I PRIKAZ FUNKCIONALNOSTI

U ovom će se poglavlju analizirati napisani kod te prikazati korištenje unutar aplikacije.

4.1. Prijava korisnika

Prilikom prvog pokretanja aplikacije, korisniku se prikazuje ekran za prijavu (Slika 4.1.). Na ovom ekranu, korisnik treba unijeti svoju adresu e-pošte i lozinku kako bi se prijavio u aplikaciju. Nakon uspješne prijave, korisnik se automatski preusmjerava na glavni zaslon aplikacije.



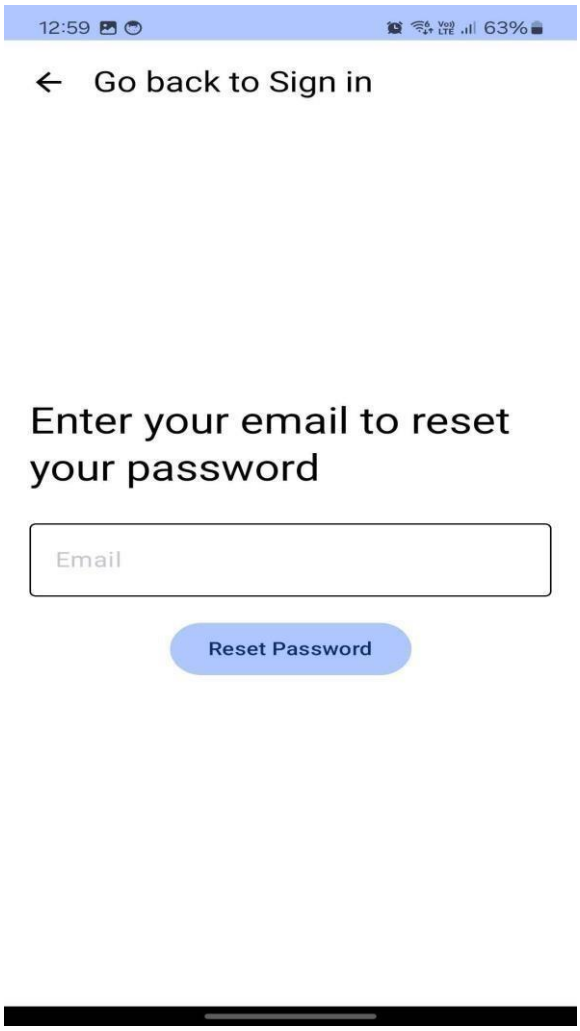
Slika 4.1. Početni ekran za prijavu.

Ekran za prijavu korisnika koristi klasu *FirebaseAuthenticationRepository* čija je zadaća implementacija funkcije za autentifikaciju korisnika korištenjem adrese e-pošte i lozinke korisnika (Slika 4.2.).

```
suspend fun signIn(email: String, password: String): Result<FirebaseUser> {
    return try {
        val result = firebaseAuth.signInWithEmailAndPassword(email, password).await()
        Result.success(result.user!!)
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

Slika 4.2. Implementacija funkcije za prijavu korisnika putem adrese e-pošte i lozinke.

U slučaju da je korisnik zaboravio lozinku za postojeći račun, pri dnu ekrana prikazanog na Slici 4.1., nalazi se poveznica *"Forgot Password?"* koja korisnika vodi na ekran za ponovno postavljanje lozinke (Slika 4.3.).



Slika 4.3. Ekran za ponovno postavljanje lozinke.

Nakon klika na gumb *"Reset password"* korisniku se šalje e-poruka s uputama za ponovno postavljanje lozinke.

```
class PasswordResetViewModel : ViewModel() {
    fun resetPassword(email: String, callback: (Boolean, String?) -> Unit) {
        val firebaseInstance = FirebaseAuth.getInstance()

        firebaseInstance.sendPasswordResetEmail(email)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    callback(true, null)
                } else {
                    callback(false, task.exception?.message)
                }
            }
    }
}
```

Slika 4.4. Implementacija logike ponovnog postavljanja lozinke.

4.2. Registracija korisnika

U slučaju da korisnik još nema račun, ispod gumba za prijavu, prikazanog na Slici 4.1., nalazi se poveznica *"Not signed up yet? Click here"*. Klikom na ovu poveznicu, korisnik se preusmjerava na ekran za registraciju gdje treba unijeti potrebne podatke za izradu računa (Slika 4.5.). Nakon uspješne registracije, korisnik se također automatski preusmjerava na glavni zaslon aplikacije.

Slika 4.5. Zaslon za registraciju korisnika.

Ekran za registraciju također koristi *FirebaseAuthenticationRepository* za implementaciju logike registracije korisnika (Slika 4.6.).

```
suspend fun signUp(email: String, password: String): Result<FirebaseUser> {
    return if (checkIfUserExists(email)) {
        Result.failure(Exception("An account with this email already exists"))
    } else {
        try {
            val result = firebaseAuth.createUserWithEmailAndPassword(email, password).await()
            Result.success(result.user!!)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

Slika 4.6. Implementacija registriranja korisnika.

Pri kreiranju računa provjerava se postoji li korisnik s navedenom adresom e-pošte pomoću funkcije *checkIfUserExists()* prikazane na Slici 4.7. Ukoliko je adresa već prije registrirana, funkcija vraća vrijednost istina, a u suprotnom vraća vrijednost laž. Ovisno o vraćenoj vrijednosti, korisniku se stvara korisnički račun ili mu aplikacija izbacuje odgovarajuću grešku.

```
suspend fun checkIfUserExists(email: String): Boolean {
    return try {
        val result = firebaseAuth.fetchSignInMethodsForEmail(email).await()
        result.signInMethods?.isNotEmpty() ?: false
    } catch (e: Exception) {
        false
    }
}
```

Slika 4.7. Implementacija provjere postojećih korisnika.

4.2.1. Pozivanje repozitorija unutar aplikacije

Zbog korištenja Jetpack Composea i prikladne *Model-View-ViewModel* (MVVM) arhitekture, za ekrane poput *Prijave* i *Registracije* potrebni su pripadajući *ViewModeli*. U *ViewModelima* nalaze se sve metode i funkcionalnosti koje se odnose na logiku tih ekrana, čime se postiže jasna podjela odgovornosti i olakšava testiranje koda.

Radi navedenoga, potrebno je imati dva odvojena *ViewModela* - za ekran prijave (Slika 4.8.) i

za ekran registracije (Slika 4.9.). Ovi će *ViewModeli* komunicirati s *FirebaseAuthenticationRepository* kako bi izvršili potrebne radnje vezane uz prijavu i registraciju korisnika.

Ovakva podjela odgovornosti između *ViewModela* i repozitorija osigurava modularnost, čitljivost i lakše testiranje koda. Također, omogućuje nam da svaki *ViewModel* brine samo o logici koja se odnosi na njegov pripadajući ekran, čime se izbjegava nepotrebna kompleksnost i olakšava održavanje aplikacije.

```
class SignInViewModel() : ViewModel() {

    private val authenticationRepository = FirebaseAuthenticationRepository(FirebaseAuth.getInstance())
    val state = MutableStateFlow<SignInState>(SignInState.Idle)

    fun signIn(email: String, password: String) {
        state.value = SignInState.Idle
        viewModelScope.launch {
            if(authenticationRepository.signIn(email,password)==Result.success(FirebaseAuth.getInstance().currentUser)){
                state.value = SignInState.Success
            }
            else{
                state.value = SignInState.Error("An error occurred")
            }

        }
    }
    fun checkIsInputValid(email: String, password: String): Boolean {
        return email.isNotEmpty() && password.isNotEmpty()
    }
}
```

Slika 4.8. Logika ViewModela za prijavu korisnika.

```
class SignUpViewModel : ViewModel() {
    private val authenticationRepository = FirebaseAuthenticationRepository(FirebaseAuth.getInstance())

    fun signUp(email: String, password: String, onSuccess: () -> Unit, onError: (String) -> Unit) {

        viewModelScope.launch {
            try {
                authenticationRepository.signUp(email, password)
                onSuccess()
            } catch (e: Exception) {
                onError(e.message ?: "An error occurred")
            }
        }
    }
}
```

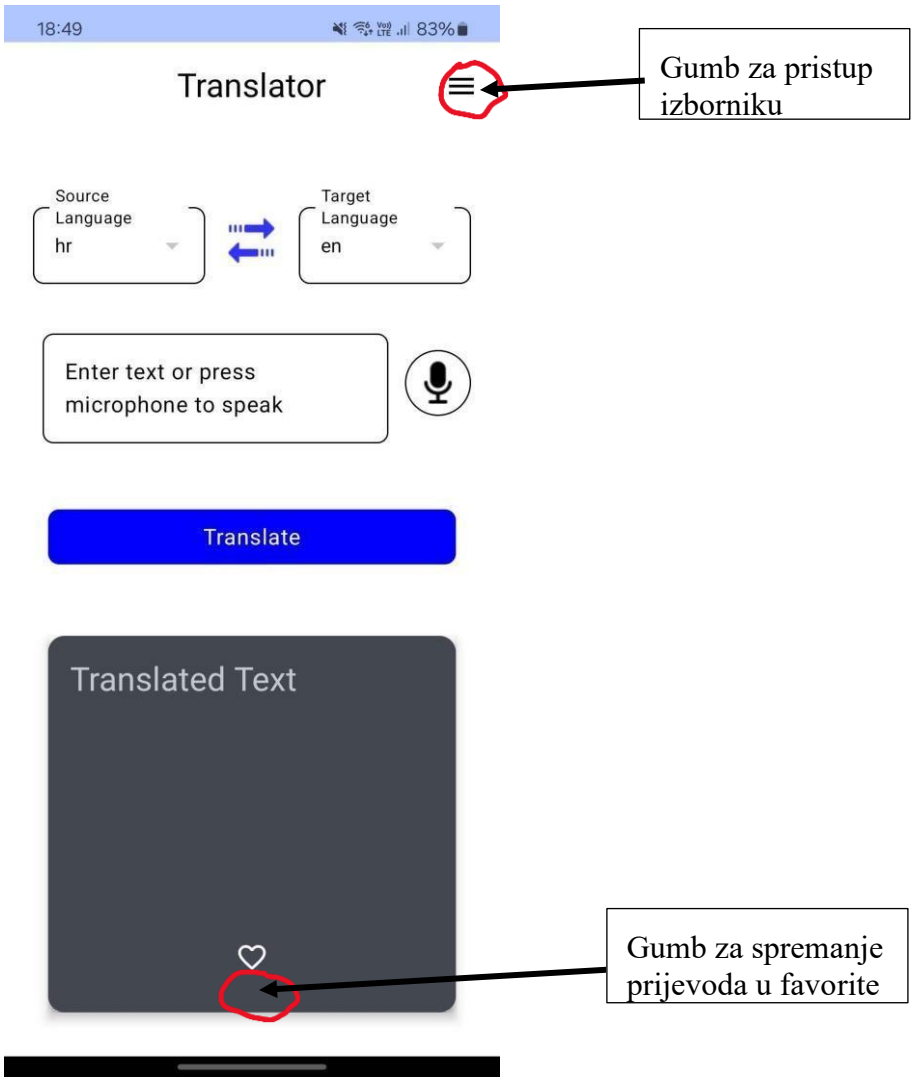
Slika 4.9. Logika ViewModela za registraciju korisnika.

Ova dva *ViewModela* unutar sebe imaju instancu *FirebaseAuthenticationRepository* klase pomoću koje pozivamo metode za prijavu i registraciju korisnika.

4.3. Glavni zaslon aplikacije

Nakon uspješne prijave, korisnik se preusmjerava na glavni zaslon aplikacije za prevođenje teksta koji se odlikuje minimalističkim dizajnom usmjerenim na temeljne elemente za brzo i učinkovito prevođenje.

Na vrhu ekrana nalaze se padajući izbornici za odabir izvornog i ciljanog jezika, s unaprijed postavljenim vrijednostima za hrvatski i engleski jezik. Klikom na strelice vrijednosti za jezike mogu se promijeniti, tako da izvorni bude engleski, a ciljani hrvatski jezik. Ispod izbornika nalazi se tekstualno polje u koje korisnik unosi tekst koji želi prevesti. Proces prevođenja pokreće se klikom na gumb *Translate,* a za one koji preferiraju unos glasom, dostupan je i gumb za aktivaciju mikrofona. Prevedeni tekst ispisuje se u odjeljku ispod, pružajući korisniku jasan i pregledan rezultat (Slika 4.10.).



Slika 4.10. Prikaz sučelja glavnog zaslona.

Osim osnovnih funkcija, ekran nudi i dodatnu mogućnost putem intuitivnog gumba koji je označen na Slici 4.10. Prvi vrhu na desnoj strani nalazi se gumb za prikaz izbornika, koji je moguće otvoriti i povlačenjem prsta s lijeva na desno po ekranu. Pri dnu se nalazi gumb u obliku srca koji služi za spremanje prijevoda ako korisnik to odluči napraviti.

4.3.1. Implementacija padajućeg izbornika

Isječak koda prikazan na Slici 4.11. prikazuje implementaciju padajućeg izbornika za odabir jezika unutar Jetpack Compose sučelja. Funkcija *LanguageDropdownMenu* prima trenutno odabrani jezik (*selectedLanguage*), funkciju za ažuriranje odabranog jezika (*onLanguageSelected*), popis dostupnih jezika (*languages*) i oznaku za prikaz iznad izbornika (*label*). Koriste se *ExposedDropdownMenuBox* i *ExposedDropdownMenu* komponente iz

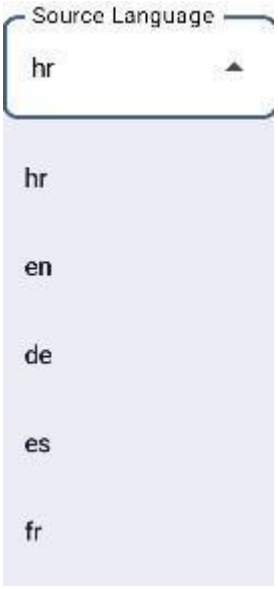
Compose Material biblioteke kako bi se stvorio interaktivni izbornik. Stanje *expanded* prati je li izbornik otvoren ili zatvoren, dok *currentSelectedLanguage* pohranjuje trenutno odabrani jezik. *TextField* komponenta prikazuje odabrani jezik, a strelica unutar komponente na desnoj strani označava da se radi o padajućem izborniku.

```
fun LanguageDropdownMenu(
    selectedLanguage: String,onLanguageSelected: (String) -> Unit,
    languages: List<String>,
    label: String
) {
    var expanded by remember { mutableStateOf( value: false) }
    var currentSelectedLanguage by remember { mutableStateOf(selectedLanguage) }
    ExposedDropdownMenuBox(expanded = expanded,
        onExpandedChange = { expanded = !expanded },
        modifier = Modifier.width(150.dp),

    ) {
        OutlinedTextField(
            readOnly = true,
            value = currentSelectedLanguage,
            onValueChange = {},
            label = { Text(label, color = Color.Black) },
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
            colors = ExposedDropdownMenuDefaults.outlinedTextFieldColors(
                focusedBorderColor = MaterialTheme.colorScheme.primary,
                unfocusedBorderColor = Color.Black,
            ),
            shape = RoundedCornerShape(8.dp),
            modifier = Modifier.menuAnchor(),
            textStyle = TextStyle(color = Color.Black),
        )
        ExposedDropdownMenu(
            expanded = expanded,
            onDismissRequest = { expanded = false }
        ) {
            languages.forEach { language ->
                DropdownMenuItem(
                    onClick = {
                        currentSelectedLanguage = language
                        onLanguageSelected(language)
                        expanded = false
                    },
                    text = { Text(language) }
                )
            }
        }
    }
}
```

Slika 4.11. Implementacija padajućeg izbornika.

Kada korisnik klikne na izbornik, popis dostupnih jezika (Slika 4.12.) prikazuje se pomoću *ExposedDropdownMenu*. Odabirom nekog jezika, poziva se *onLanguageSelected* funkcija kako bi se ažurirao odabrani jezik, a izbornik se zatvara. Ovaj kod osigurava lagan i jednostavan način odabira jezika unutar korisničkog sučelja aplikacije.



Slika 4.12. Prikaz padajućeg izbornika za izbor jezika.

Klikom na padajući izbornik prikazuju se jezici na i s kojih je moguće prevoditi. Trenutno su dostupni jezici hrvatski, engleski, njemački, španjolski i francuski jer su najkorišteniji na području Europe. Međutim, moguće je dodavanje jezika u budućnosti.

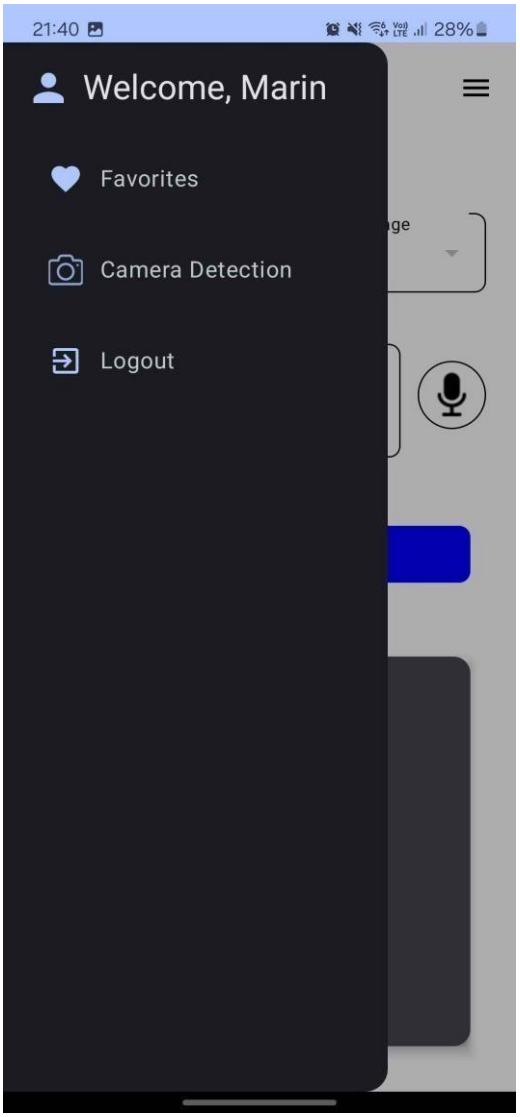
4.3.2. Prikaz bočnog izbornika

Bočni izbornik u aplikaciji omogućuje korisniku brz pristup ključnim informacijama i funkcijama. Aktiviran povlačenjem prsta s lijeve strane ekrana ili klikom na ikonu izbornika, ovaj se izbornik sastoji od nekoliko elemenata. Na vrhu se nalazi personalizirani pozdrav s imenom korisnika stvarajući osjećaj individualiziranog pristupa.

Opcija *Favorites* vodi korisnika do pregleda i upravljanja prethodno sačuvanim prijevodima nudeći praktičan način za ponovnu upotrebu sadržaja. Prijevodi su spremljeni u potpunosti; prvo je prikazan izvorni tekst, a zatim prijevod. Spremljeni prijevodi nude korisniku brz pristup izrazima s kojima bi se mogli češće susresti, na primjer pri naručivanju nečega, uputa za smjer.

Nakon toga nalazi se opcija *Camera Detection* koja prvo provjerava je li se dopustila mogućnost korištenja kamere. Nakon dopuštanja korištenja kamere korisnik se prebacuje na ekran namijenjen za prijevod teksta sa slika, što će biti objašnjeno u poglavlju 4.6.

Na kraju, opcija *Logout* omogućuje korisniku sigurno izlaženje iz aplikacije čuvajući privatnost i sigurnost podataka. Ovaj izbornik služi kao središnje mjesto za navigaciju i personalizaciju unutar aplikacije (Slika 4.13.).

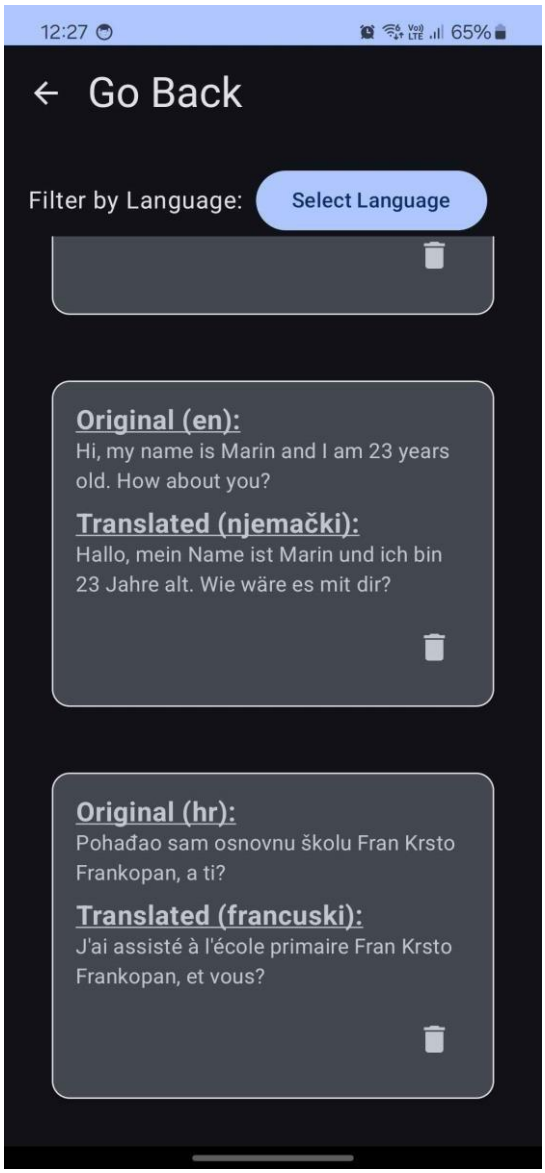


Slika 4.13. Prikaz bočnog izbornika.

4.4. Zaslon sa spremljenim prijevodima

Zaslon na Slici 4.14. omogućuje pregled i upravljanje spremljenim prijevodima nudeći praktičan način lako dostupnih često korištenih prijevoda. Minimalistički dizajn sučelja osigurava brzo i jednostavno snalaženje, a istovremeno potiče učinkovitost u komunikaciji i prevođenju.

Na vrhu ekrana nalazi se ikona strelice koja služi za povratak na glavni ekran aplikacije. Ispod ikone strelice, korisniku se prikazuje popis svih spremljenih prijevoda. Svaki prijevod popraćen je ikonom kante za smeće kojom korisnik može ukloniti neželjene prijevode s popisa. Ova funkcionalnost omogućuje korisniku personaliziranje iskustva rada s aplikacijom i prilagođavanje specifičnim potrebama, čime se povećava ukupna učinkovitost i praktičnost aplikacije.



Slika 4.14. Izgled zaslona sa spremljenim prijevodima.

4.4.1. Funkcionalnosti zaslona sa spremljenim prijevodima

U aplikaciji se koristi *Firebase Cloud Firestore*, usluga za pohranu podataka u oblaku, kako bi se omogućilo jednostavno i učinkovito spremanje te dohvaćanje korisničkih podataka poput spremljenih prijevoda. *Cloud Firestore* eliminira potrebu za pisanjem SQL koda, čime se pojednostavljuje razvoj aplikacije.

Za interakciju s *Cloud Firestoreom* koristi se poseban repozitorij, *FavoritesRepository*, koji sadrži sve potrebne funkcije za dohvaćanje, spremanje i brisanje podataka o spremljenim prijevodima. *ViewModel* za ekran favorita (*FavoritesViewModel*) komunicira s ovim repozitorijem

kako bi korisniku prikazao popis spremljenih prijevoda te omogućio dodavanje novih ili brisanje postojećih prijevoda prikazanih Slikom 4.15.

```
sealed class FavoritesState {
    object Loading : FavoritesState()
    data class Error(val message: String) : FavoritesState()
    data class Success(val state: FavoriteTranslations?) : FavoritesState()
}

class FavoritesViewModel(
    private val repository: FavoritesRepository = FavoritesRepository()
) : ViewModel() {
    init {
        viewModelScope.launch {
            getFavorites()
        }
    }

    val state = MutableStateFlow<FavoritesState>(FavoritesState.Loading)

    fun saveToFavorites(item: TranslationItem) {
        viewModelScope.launch {
            repository.saveTextToFirestore(item.text, item.languageCode,item.originalLanguage, item.originalText)
            val updatedFavorites = repository.getFavorites()
            state.value = FavoritesState.Success(updatedFavorites)
        }
    }

    private suspend fun getFavorites() {
        val result = repository.getFavorites()
        state.value = FavoritesState.Success(result)
    }

    fun deleteFavorite(item: TranslationItem, context: Context) {
        viewModelScope.launch {
            repository.deleteFavoriteFromFirestore(item, context)
            val updatedFavorites = repository.getFavorites()
            state.value = FavoritesState.Success(updatedFavorites)
        }
    }
}
```

Slika 4.15. Logika FavoritesViewModela.

4.4.2. Repozitorij omiljenih sadržaja

Repozitorij *FavoritesRepository* (Slika 4.16.) je ključna komponenta u aplikaciji za prevođenje koja omogućava korisnicima da spremaju, dohvaćaju i brišu svoje omiljene prijevode. Koristi se *Firebase Firestore*, NoSQL baza podataka u oblaku za kontinuirano pohranjivanje ovih prijevoda. Ovaj je repozitorij dizajniran za jednostavno korištenje i lako integriranje s *ViewModelom* ekrana omiljenih sadržaja pružajući robusno i pouzdano rješenje za upravljanje omiljenim prijevodima korisnika.

```
class FavoritesRepository {

    fun saveTextToFirestore(text: String, languageCode: String, originalLanguage: String, originalText: String,
    ) {
        val firebaseAuth: FirebaseAuth = FirebaseAuth.getInstance()
        val currentUser = firebaseAuth.currentUser
        val userId = currentUser?.uid

        if (userId != null) {
            val translationItem = hashMapOf(
                "originalText" to originalText,
                "originalLanguage" to originalLanguage,
                "text" to text,
                "languageCode" to languageCode
            )
            val userDocumentRef = database.collection( collectionPath: "users").document(userId)
            userDocumentRef.update( field: "favoritedTranslations", FieldValue.arrayUnion(translationItem))
                .addOnSuccessListener {
                    state.value = FavoritesState.Success( state: null)
                }
                .addOnFailureListener {
                    state.value = FavoritesState.Error("Error favoriting translation")
                }
        }
    }
    suspend fun getFavorites(): FavoriteTranslations? {
        val firebaseAuth: FirebaseAuth = FirebaseAuth.getInstance()
        val currentUser = firebaseAuth.currentUser
        val userId = currentUser?.uid
        val data = userId?.let { database.collection( collectionPath: "users").document(it).get().await() }
        return (data as DocumentSnapshot).toObject(FavoriteTranslations::class.java)
    }
    fun deleteFavoriteFromFirestore(translationItem: TranslationItem, context: Context) {
        val firebaseAuth = FirebaseAuth.getInstance()
        val currentUser = firebaseAuth.currentUser
        val userId = currentUser?.uid

        if (userId != null) {
            val userDocRef = database.collection( collectionPath: "users").document(userId)
            val itemToDelete = hashMapOf(
                "originalText" to translationItem.originalText,
                "originalLanguage" to translationItem.originalLanguage,
                "text" to translationItem.text,
                "languageCode" to translationItem.languageCode
            )
            userDocRef.update( field: "favoritedTranslations", FieldValue.arrayRemove(itemToDelete))
                .addOnSuccessListener {
                    state.value = FavoritesState.Success( state: null)
                }
                .addOnFailureListener {
                    state.value = FavoritesState.Error("Error deleting from favorites")
                }
        }
```

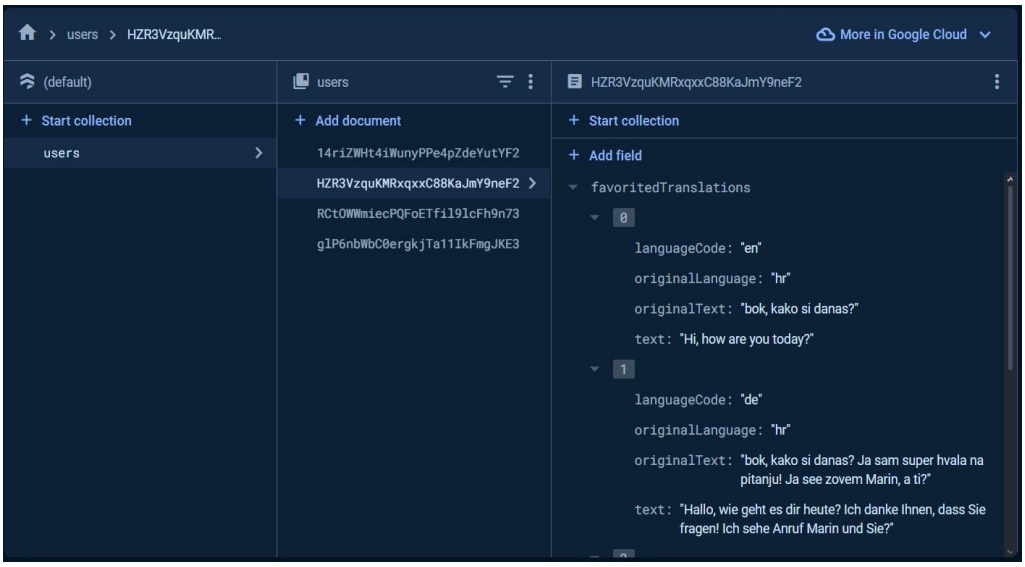
Slika 4.16. Implementacija Favorites repozitorija.

Funkcija *saveTextToFirestore* sprema novi prijevod u *Firestore* kolekciju *users* pod dokumentom koji odgovara jedinstvenom ID-u korisnika. Prije spremanja, provjerava se je li korisnik prijavljen kako bi se osigurala sigurnost i privatnost podataka. Funkcija *getFavorites* dohvaća sve spremljene prijevode za trenutno prijavljenog korisnika. Ako korisnik nije prijavljen, vraća *null*, čime se sprječava neovlašten pristup podacima. Na kraju, funkcija

deleteFavoriteFromFirestore omogućuje uklanjanje određenih prijevoda iz popisa favorita. Slično kao i kod spremanja, provjerava se je li korisnik prijavljen prije nego što se izvrši brisanje. Sve implementacije funkcija prikazane su Slikom 4.16.

4.4.3. Izgled baze podataka

Baza podataka za ovu aplikaciju sastoji se od kolekcije *users* koja sprema dokumente za svakog pojedinog korisnika. Unutar svakog korisničkog dokumenta nalazi se polje *FavoritedTranslations*. Ovo polje je niz koji sadrži spremljene prijevode korisnika. U prikazanom primjeru, korisnik je spremio pet prijevoda na različitim jezicima (hrvatski, njemački, francuski i engleski). Izgled baze podataka prikazan je na Slici 4.17. Ovakva struktura podataka omogućuje jednostavno spremanje i dohvaćanje omiljenih prijevoda za svakog korisnika. Kada korisnik označi prijevod kao omiljen, aplikacija dodaje taj prijevod u niz *FavoritedTranslations* unutar njegovog korisničkog dokumenta. Kada korisnik želi vidjeti svoje omiljene prijevode, aplikacija dohvaća podatke iz tog polja.



Slika 4.17. Struktura baze podataka.

4.4.4. Pozivanje repozitorija

FavoritesViewModel (Slika 4.18.) ima ulogu u upravljanju podacima o omiljenim prijevodima korisnika unutar aplikacije. Koristi se arhitektura MVVM kako bi se osigurala jasna podjela odgovornosti i učinkovito komuniciranje s repozitorijem podataka *FavoritesRepository.*

```
sealed class FavoritesState {
    object Loading : FavoritesState()
    data class Error(val message: String) : FavoritesState()
    data class Success(val state: FavoriteTranslations?) : FavoritesState()
}

class FavoritesViewModel(
    private val repository: FavoritesRepository = FavoritesRepository()
) : ViewModel() {
    init {
        viewModelScope.launch {
            getFavorites()
        }
    }

    val state = MutableStateFlow<FavoritesState>(FavoritesState.Loading)

    fun saveToFavorites(item: TranslationItem) {
        viewModelScope.launch {
            repository.saveTextToFirestore(item.text, item.languageCode,item.originalLanguage, item.originalText)
            val updatedFavorites = repository.getFavorites()
            state.value = FavoritesState.Success(updatedFavorites)
        }
    }

    private suspend fun getFavorites() {
        val result = repository.getFavorites()
        state.value = FavoritesState.Success(result)
    }

    fun deleteFavorite(item: TranslationItem, context: Context) {
        viewModelScope.launch {
            repository.deleteFavoriteFromFirestore(item, context)
            val updatedFavorites = repository.getFavorites()
            state.value = FavoritesState.Success(updatedFavorites)
        }
    }
}
```

Slika 4.18. Izgled ViewModela favorita.

Prilikom inicijalizacije, *FavoritesViewModel* pokreće *korutinu* koja automatski dohvaća spremljene prijevode korisnika iz *Firebase Firestore* baze podataka. Stanje podataka prati se pomoću *MutableStateFlow* objekta koji može biti u tri stanja - *Loading* (podaci se učitavaju), *Success* (podaci uspješno dohvaćeni) ili *Error* (došlo je do pogreške).

Implementacija *korutina* osigurava da se operacije dohvaćanja, spremanja i brisanja podataka izvode u pozadini, bez blokiranja glavne niti aplikacije, čime se održava responzivnost korisničkog sučelja.

FavoritesViewModel nudi funkcije *saveToFavorites* i *deleteFavorite*, koje omogućuju spremanje novih prijevoda na popis favorita i brisanje postojećih. Obje funkcije pokreću *korutine* koje komuniciraju s repozitorijem, a nakon izvršenja akcije (spremanje ili brisanje) osvježavaju popise prijevoda i ažuriraju stanje kako bi korisničko sučelje prikazalo najnovije podatke.

4.5. Korištenje audio prijevoda

Klasa *SpeechToText*, prikazana na Slici 4.19., korištena je za detekciju govora korisnika kojeg prevodi u obliku teksta. Koristi Googleovu izvornu detekciju govora koja se nalazi unutar Android operacijskog sustava.

```
class SpeechToText(
    private val language: String,
    private val context: Context,
    private val onTextRecognized: (String) -> Unit
) : RecognitionListener {

    private val speechRecognizer: SpeechRecognizer by lazy {
        SpeechRecognizer.createSpeechRecognizer(context)
    }

    init {
        speechRecognizer.setRecognitionListener(this)
    }
```

Slika 4.19. Klasa SpeechToText.

Pri uspješnoj detekciji govora, govor se pretvara u tekstni oblik funkcijom *onResults,* prikazanog na slici 4.20., te se šalje ostatku aplikacije u obliku liste znakova.

```
override fun onResults(results: android.os.Bundle?) {
    val matches = results?.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION)
    matches?.let {
        if (it.isNotEmpty()) {
            onTextRecognized.invoke(it[0])
        }
    }
}
```

Slika 4.20. Implemetacija onResults funkcije.

Klasa prima jezik koji treba prepoznati i lambda funkciju *OnTextRecognized* koja će se pozvati kada govor bude uspješno prepoznat i pretvoren u tekst. Unutar klase nalaze se funkcije za početak slušanja govora (Slika 4.21.) i funkcija koja automatski prekida slušanje nakon što je korisnik sve izrekao (Slika 4.22.). Također, postoje jednostavnije funkcije koje služe za obavještavanje korisnika da može početi govoriti i da je prestao govoriti.

```
fun startListening() {
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
        putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        putExtra(RecognizerIntent.EXTRA_LANGUAGE, language)
    }
    speechRecognizer.startListening(intent)
}
```

Slika 4.21. Funkcija za početak slušanja govora.

U ovoj funkciji prvo se stvara *Intent* objekt s akcijom *RecognizerIntent.ACTION_RECOGNIZE_SPEECH*, što signalizira sustavu da treba započeti slušanje i prepoznavanje govora. Ovaj *Intent* je konfiguriran unutar bloka *apply*, koji omogućuje dodavanje dodatnih parametara izravno prilikom kreiranja *Intent* objekta.

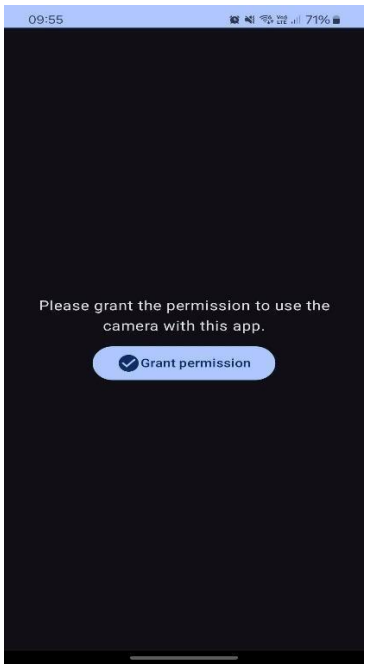
```
override fun onEndOfSpeech() {
    // Called when the user has finished speaking.
    stopListening()
    Toast.makeText(
        context,
        text: "Stopped speaking",
        Toast.LENGTH_SHORT
    ).show()
}
```

Slika 4.22. Funkcija za prekidanje slušanja.

Ova funkcija prepisuje metodu *onEndOfSpeech* iz sučelja *RecognitionListener*. To znači da ova funkcija zamjenjuje zadano ponašanje kako bi pružila specifične upute koje će se izvršiti kad korisnik prestane govoriti. U ovom slučaju, prekida se slušanje, a korisnik dobiva obavijest da slušanje govora više nije aktivno.

4.6. Zaslon za detekciju teksta kamerom

Prilikom pristupa ekranu za detekciju kamerom, ako aplikaciji nije dopušteno korištenje kamere, korisnik se preusmjerava na ekran na kojem se traži dopuštenje za korištenje kamere (Slika 4.23.) kako bi se mogla aktivirati mogućnost detekcije s kamerom.



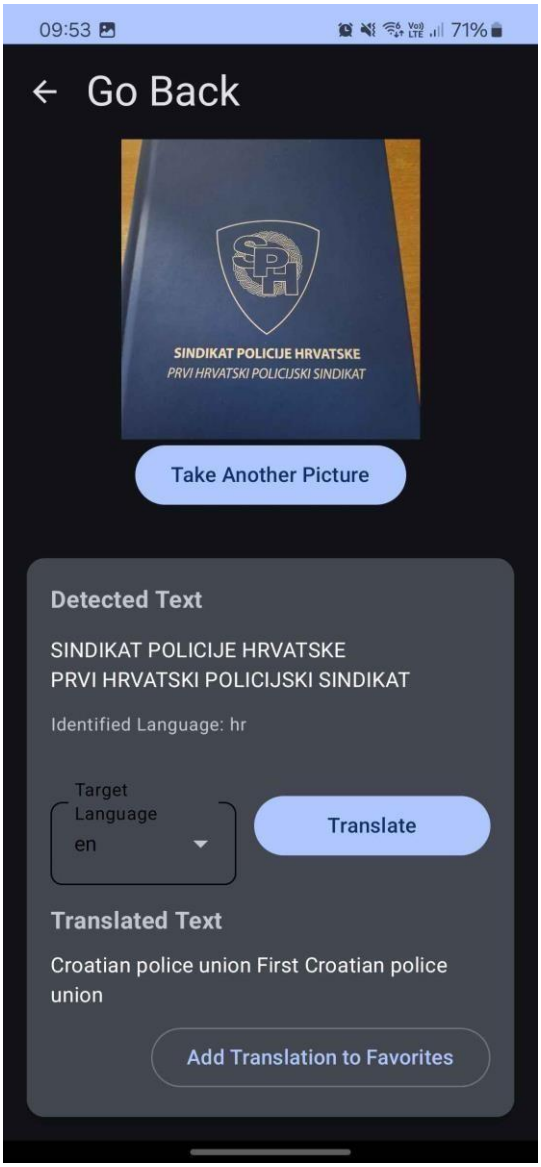
Slika 4.23. Ekran za traženje pristupa kameri.

Nakon uspješnog dopuštenja korisnik je prebačen na ekran za detekciju teksta sa slika prikazanog na slici 4.24.

Slika 4.24. Ekran za detekciju teksta sa slika.

Na ekranu je dostupan prostor s kamerom koji korisniku omogućava da vidi što slika. Na sredini navedenog prostora nalazi se gumb za spremanje slike koja se prvo mora spremiti da bi se tekst mogao detektirati. Nakon spremanja slike prikazuje se prepoznati tekst, ako isti postoji, u za

to navedenom prostoru. Također, prikazuje se prepoznati jezik. Ispod navedenog nalazi se izbornik za izbor ciljanog jezika na koji želimo prevesti prepoznati tekst. Pritiskom gumba *Translate* prikazuje se pripadajući prijevod prepoznatoga teksta, što je prikazano na Slici 4.25.



Slika 4.25. Prikaz funkcije ekrana za detekciju teksta sa slika.

Pritiskom na gumb *Translate* uz prijevod se prikazuje i opcija spremanja prijevoda u favorite, no slika se ne sprema. Gumb za spremanje slike se mijenja u gumb za spremanje nove slike. Nakon pritiska na isti slika se briše i korisniku se pruža mogućnost spremanja nove slike.

4.7. Implementacija detekcije sa slika

Za spremanje privremenih slika te upravljanja kamerom korištena je biblioteka CameraX. Za spremanje slika zaslužna je funkcija *captureImage* prikazana na slici 4.26.

```
fun captureImage() {
    Log.d( tag: "CameraContent", msg: "Capturing image")
    val tempFile = File.createTempFile( prefix: "temp_image", suffix: ".jpg", context.cacheDir).apply {
        deleteOnExit()
    }
    val outputOptions = ImageCapture.OutputFileOptions.Builder(tempFile).build()
    cameraController.takePicture(
        outputOptions,
        ContextCompat.getMainExecutor(context),
        object : ImageCapture.OnImageSavedCallback {
            override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
                outputFileResults.savedUri?.let { uri ->
                    capturedImageUri = uri
                    isPictureTaken = true
                    coroutineScope.launch {
                        textRecognitionAnalyzer.analyzeImage(uri)
                    }
                }
            }
            override fun onError(exception: ImageCaptureException) {
                Log.e( tag: "CameraContent", msg: "Image capture failed", exception)
                detectedText = "Failed to capture image"
            }
        }
    )
}
```

Slika 4.26. Implementacija funkcije captureImage.

Funkcija prvo kreira privremenu .jpg datoteku u *cache* direktoriju aplikacije. Navedena će se datoteka koristiti za pohranu spremljene slike. Funkcija *deleteOnExit* osigurava da je ista obrisana pri izlasku iz aplikacije. Nakon toga, definirane su opcije za snimanje slika poslije kojeg se na kameri poziva funkcija *takePicture* s navedenim opcijama kamere. Zatim funkcija *takePicture* sprema sliku, postavlja joj putanju u direktorij u kojemu će se nalaziti i poziva *textRecognitionAnalyzer.analyzeImage* koja kao argument prima putanju do slike. Nakon spremanja slike i ponovnog zahtjeva za spremanja nove slike poziva se funkcija *retakePicture* koja briše prijašnju sliku te prijašnje prijevode i prepoznati tekst (Slika 4.27).

```
fun retakePicture() {
    capturedImageUri?.let { uri ->
        val fileName = uri.path?.substringAfterLast( delimiter: "/")
        val file = fileName?.let { File(context.cacheDir, it) }
        file?.delete()

    }
    capturedImageUri = null
    isPictureTaken = false
    detectedText = "No text detected yet.."
    identifiedLanguageState = ""
    showTranslation = false
}
```

Slika 4.27. Funkcija retakePicture.

Za prikaz slike prvo se provjerava je li slika spremljena. Ako je, slika se prikazuje pomoću *AsyncImage* koji učinkovito učitava i prikazuje spremljenu sliku na temelju zadane putanje slike. U suprotnom, ako nema spremljene slike, prikazuje se pregled kamere koristeći *AndroidView*. Unutar *AndroidView* stvara se *PreviewView* za prikazivanja pregleda kamere te se postavlja *cameraController* za upravljanje kamerom uređaja (Slika 4.28).

```
if (isPictureTaken && capturedImageUri != null) {
    key(capturedImageUri) {
        AsyncImage(
            model = capturedImageUri,
            contentDescription = "Captured image",
            modifier = Modifier.fillMaxSize(),
            contentScale = ContentScale.Fit
        )
        DisposableEffect(capturedImageUri) {
            onDispose {
                deleteImageUri = capturedImageUri
            }
        }
    }
} else {
    Log.d( tag: "CameraContent", msg: "No captured image")
    AndroidView(
        factory = { ctx ->
            PreviewView(ctx).apply {
                this.controller = cameraController
            }
        },
        modifier = Modifier.fillMaxSize()
    )
}
```

Slika 4.28. Prikaz pregleda kamere i spremljene fotografije.

Nakon uspješnog spremanja slike i pozivanja *textRecognitionAnalyzer.analyzeImage* unutar funkcije *takePicture,* slika se prosljeđuje funkciji za detekciju teksta sa navedene slike. Funkcija *analyzeImage* prima kao argument lokaciju spremljene slike pomoću koje se učitava slika te se na

istoj detektira tekst. Nakon uspješne detekcije teksta, to jest ako postoji tekst na slici, funkcija vraća listu teksta (Slika 4.29).

```
suspend fun analyzeImage(uri: Uri) {
    try {
        val inputImage = withContext(Dispatchers.IO) {
            InputImage.fromFilePath(context, uri)
        }
        val visionText = textRecognizer.process(inputImage).await()
        val detectedText = visionText.text

        if (detectedText.isNotBlank()) {
            val identifiedLanguage = languageIdentifier.identifyLanguage(detectedText).await()
            onTextRecognized(detectedText, identifiedLanguage)
        }
    } catch (e: Exception) {
        Log.e( tag: "TextRecognitionAnalyzer",  msg: "Error processing image", e)
        onTextRecognized("Error processing image", "Unknown")
    }
}
```

Slika 4.29. Implementacija funkcije analyzeImage.

5. ZAKLJUČAK

Cilj je ovog završnog rada bio izrada mobilne aplikacije za prevođenje putem teksta i govora. Nakon provedenog istraživanja postojećih rješenja, zaključeno je da iako postoji mnoštvo aplikacija za prevođenje, rijetke su one koje kombiniraju prevođenje s mogućnošću personalizacije korisničkog računa i spremanje prijevoda.

Za potrebe razvoja aplikacije dostupne na Android platformi korišten je Android Studio s programskim jezikom Kotlin i modernim alatom za korisničko sučelje Jetpack Compose. Ova tehnologija osigurala je razvoj moderne i responzivne aplikacije. Firebase je implementiran za autentifikaciju korisnika i pohranu podataka, što je omogućilo sigurno upravljanje korisničkim računima i jednostavno spremanje prijevoda u oblak.

U funkcionalnom pogledu, aplikacija nudi sveobuhvatno rješenje za prevođenje. Korisnici mogu unositi tekst za prevođenje ili koristiti glasovni unos, što aplikaciju čini pristupačnom u različitim situacijama. Mogućnost spremanja prijevoda posebno je korisna za često korištene fraze ili učenje jezika. Sustav prijave i odjave korisnika omogućuje personalizirano iskustvo i sigurnost podataka.

Ovaj završni rad predstavlja sveobuhvatno rješenje za svakodnevne potrebe prevođenja, kombinirajući funkcionalnost s personaliziranim korisničkim sustavom. Aplikacija, razvijena korištenjem modernih tehnologija i pristupa usmjerenog na korisnika, nudi učinkovito i prilagodljivo rješenje za prevođenje. Iako je aplikacija uspješno implementirana, mogući su i daljnji koraci za unapređenje, poput proširenja broja podržanih jezika, poboljšana preciznost prepoznavanja govora te integracija s drugim jezičnim alatima, čime aplikacija može postati još korisnija i relevantnija.

LITERATURA

[1] T.Poibeau, Machine Translation, MIT Press, USA, 2017.

[2] N., Sethiya, C. K., Maurya, „End-to-End Speech-to-Text Translation: A Survey“. arXiv, 08lip-2024.

[3] „ICT Business | Google Translate sada prevodi na više od 100 jezika i pokriva 99 posto online populacije“ [online]. Dostupno na: https://www.ictbusiness.info/internet/googletranslatesada-prevodi-na-vise-od-100-jezika-i-pokriva-99-posto-online-populacije. [Pristupljeno: 11.9.2024.].

[4] „Google prevoditelj“, *Wikipedija*. 01-sij-2022.

[5] „Microsoft Translator“, *Wikipedia*. 30-kol-2024.

[6] „Dictionary and online translation - Yandex Translate.“ [online]. Dostupno na: https://translate.yandex.com/en/translator/English-Russian. [Pristupljeno: 11.9.2024.].

[7] „Naver Papago“, *Wikipedia*. 03-kol-2024.

[8] „DeepL Translate: The world’s most accurate translator“ [online]. Dostupno na: https://www.deepl.com/translator. [Pristupljeno: 11.9.2024.].

[9] „DeepL Translator“, *Wikipedia*. 23-kol-2024.

[10] „Android | Do More With Google on Android Phones & Devices“ [online]. Dostupno na: https://www.android.com/. [Pristupljeno: 11.9.2024.].

[11] „Download Android Studio & App Tools“ [online]. Dostupno na: https://developer.android.com/studio. [Pristupljeno: 11.9.2024.].

[12] „Kotlin Programming Language“ [online]. Dostupno na: https://kotlinlang.org/. [Pristupljeno: 11.9.2024.].

[13] „Jetpack Compose UI App Development Toolkit“ [online]. Dostupno na: https://developer.android.com/compose. [Pristupljeno: 11.9.2024.].

[14] „Firebase | Google’s Mobile and Web App Development Platform“ [online]. Dostupno na: https://firebase.google.com/. [Pristupljeno: 11.9.2024.].

[15] „Firestore“ [online]. Dostupno na: https://firebase.google.com/docs/firestore. [Pristupljeno: 11.9.2024.].

[16] „Firebase Authentication“ [online]. Dostupno na: https://firebase.google.com/docs/auth. [Pristupljeno: 11.9.2024.].

[17] „ML Kit | Google for Developers“ [online]. Dostupno na: https://developers.google.com/ml-kit. [Pristupljeno: 11.9.2024.].

[18] „CameraX overview | Android media“ [online]. Dostupno na: https://developer.android.com/media/camera/camerax. [Pristupljeno: 11.9.2024.].

SAŽETAK

Ovaj završni rad prikazuje razvoj mobilne aplikacije za detekciju teksta sa slike te prevođenje teksta i govora. Prilikom istraživanja postojećih rješenja utvrđeno je da unatoč velikom broju aplikacija za prevođenje, rijetke omogućuju kombinaciju spremanja prijevoda i personalizacije, što aplikacija izrađena za potrebe ovog završnog rada nudi. Namijenjena je za korištenje na Android platformi operacijskog sustava, a za njenu su se izradu koristile moderne tehnologije poput Android Studija, programskog jezika Kotlin i Jetpack Composea za izradu modernog korisničkog sučelja. Koristile su se i usluge Firebasea za autentifikaciju i pohranu podataka te Google ML Kit za strojno učenje. Aplikacija predstavlja sveobuhvatno rješenje za jezične potrebe sporazumijevanja s mogućnostima proširenja broja podržanih jezika i poboljšanja preciznosti prepoznavanja govora.

Ključne riječi: detekcija teksta sa slike, mobilna aplikacija, personalizacija prijevoda, pretvaranje govora u tekst, prevođenje teksta

ABSTRACT

MOBILE APLICATION FOR TEXT AND SPEECH TRANSLATION

This final paper aims to develop a mobile text-to-speech translation as well as text detection application. When researching existing solutions, it was found that despite the large number of translation applications, few allow a combination of translations and personalization. The application offers the possibility of personalizing by using a user account and saving translations. It is designed to be used on the Android operating system platform. Modern technologies such as Android Studio, Kotlin programming language and Jetpack Compose were used to develop the application. Furthermore, Firebase services for authentication and data storage, and Google ML Kit for machine learning were also used in the development process. The application represents a comprehensive solution for the language needs of communication with the possibility of expanding the number of supported languages and improving the precision of speech recognition.

Key words: text detection from an image, mobile application, personalization of translations, speech-to-text conversion, text translation