Mobilna aplikacija za kreiranje radnoga naloga

Budak, Marko

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:092239

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-17**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

Mobilna aplikacija za kreiranje radnoga naloga

Završni rad

Marko Budak

Osijek, 2023.

SADRŽAJ

1. UVOD 4

1.1 Zadatak završnog rada 4

2. USPOREDBA S POSTOJEĆIM APLIKACIJAMA 5

2.1 Timesheet- Work Hours Tracker 5

2.2 Timesheet: Work Hours Tracker 5

2.3 MaintainX Work Order CMMS 6

3. RAZVOJNO OKRUŽENJE 7

3.1 Android Studio 7

3.2 Programski jezik Kotlin 8

4. KORIŠTENE TEHNOLOGIJE 9

4.1 Firebase 9

4.1.2 Firebase autentifikacija korisnika 10

4.1.3 Firebase Firestore 10

5. PROGRAMSKO RJEŠENJE ZA IZRADU APLIKACIJE 12

5.1. Početni zaslon 12

5.2. Registracija korisnika 13

5.3. Prijava korisnika 15

5.4. Resetiranje lozinke 16

5.5. Kreiranje radnoga naloga 17

5.6. Spremanje ili brisanje radnoga naloga 23

5.7. Kolekcija radnih naloga 26

6. IZGLED APLIKACIJE 28

6.1. Početni zaslon 28

6.2. Zaslon za registraciju korisniku 29

6.3. Zaslon za prijavu 30

6.4. Zaslon za resetiranje lozinke 31

6.5. Zaslon za kreiranje radnoga naloga 32
6.5. Zaslon za spremanje ili brisanje radnoga naloga 34
6.6. Zaslon kolekcije radnog naloga 35
7. ZAKLJUČAK 36
LITERATURA 37
SAŽETAK 38
ABSTRACT 39

1. UVOD

Postojeći procesi kreiranja radnih naloga često su složeni, vremenski zahtjevni i podložni pogreškama. Ručno popunjavanje papirnatih obrazaca ili korištenje neadekvatnih digitalnih alata dovode do gubitka vremena, smanjenja produktivnosti i potencijalnih pogrešaka. Ova mobilna aplikacija ima za cilj riješiti te probleme pružajući intuitivno sučelje za brzo i jednostavno kreiranje radnih naloga putem mobilnih uređaja. Automatizacija procesa, uključujući unos podataka, dodjelu zaduženja i generiranje izvještaja, omogućuje korisnicima da uštede vrijeme i minimiziraju pogreške.

Prvo poglavlje obuhvaća usporedbu mobilne aplikacije s već postojećim primjerima aplikacije na istu temu. Drugo poglavlje daje teoretsku podlogu razvojnog okruženja kreiranja aplikacije. Treće poglavlje opisuje i objašnjava sve korištene tehnologije. Četvrto poglavlje opisuje način razvoja i programsko rješenje aplikacije dok peto i posljednje poglavlje prikazuje izgled mobilne aplikacije gdje je svaki zaslon prikazan i objašnjen pomoću slike i njenoga opisa.

1.1 Zadatak završnog rada

Potrebno je napraviti mobilnu aplikaciju gdje će korisnik moći odabrati posao koji će raditi. Svaki posao može imati svoje pod poslove. Poslije odabira posla postoji mogućnost odabira materijala i alata koji će se koristiti. Nakon odabira automatski se pokreće radni nalog tako da se zabilježi vrijeme početka. Omogućiti unos pauze ili bilo kakvog zastoja u radu. Nakon odabira završetka rada bilježi se vrijeme završetka. Tako formirani radni nalog sprema se u odgovarajući format JSON ili XML i šalje se na odgovarajući server firme. Postavke, ime prezime i firmu treba omogućiti za unos kod prvog korištenja aplikacije.

2. USPOREDBA S POSTOJEĆIM APLIKACIJAMA

Zbog velike upotrebe radnih naloga u svakodnevnom životu, postoji mnogo gotovih rješenja za kreiranje radnoga naloga na internetu. No, što čini aplikacije posebnima je način na koji korisnik unosi ili odabire željene opcije te kako i gdje se spremaju radni nalozi korisnika. Na Trgovini Play i App Store-u, neke od najpopularnijih aplikacija na temu kreiranje radnoga naloga su: Timesheet – Work Hours Tracker, Timesheet: Work Hours Tracker i MaintainX Work Order CMMS.

2.1 Timesheet- Work Hours Tracker

Timesheet - Work Hours Tracker, dostupna u [1], je mobilna aplikacija koja omogućuje korisnicima praćenje i upravljanje radnim vremenom. Ova aplikacija omogućuje korisnicima da jednostavno bilježe početak i završetak radnog vremena, prate vrijeme utrošeno na različite projekte i zadatke te generiraju izvješća o produktivnosti. Korisnici mogu postaviti cijene za projekte i zadatke te izračunati ukupni iznos koji se treba fakturirati. Aplikacija također može omogućiti integraciju s drugim alatima i platformama kako bi se olakšala sinkronizacija i koordinacija. Aplikacija pomaže korisnicima u učinkovitom praćenju vremena rada, upravljanju projektima i zadacima te olakšava administrativne zadatke kao što su izvješća i fakturiranje. Postoje dva načina korištenja, besplatni i plaćeni. Besplatni način korištenja korisniku omogućuje pravljenje određenoga broja radnih naloga nakon kojih mora preći na plaćenu verziju aplikacije.

2.2 Timesheet: Work Hours Tracker

Timesheet: Work Hours Tracker, dostupna u [2], je aplikacija tvrtke Timechief kojoj je također cilj omogućiti jednostavno pravljenje radnih naloga . Aplikacija ima brojne mogućnost poput automatskog upisa dnevnog ili tjednog prekovremenog rada, praćenje provedenog vremena radeći ili na pauzi, spremanje i dijeljenje radnih naloga pomoću Dropbox, e-pošte ili Google diska. Aplikacija nema prijavu, već se podaci spremaju lokalno. Aplikacija je besplatna, no unutar aplikacije postoji mogućnost prelaska na plaćenu verziju gdje korisnik ima više mogućnosti.

2.3 MaintainX Work Order CMMS

MaintainX, dostupan u [3], je softversko rješenje za upravljanje radnim nalozima. Pruža korisnicima mogućnost digitalizacije procesa održavanja, olakšavajući izradu, upravljanje i praćenje radnih naloga. Aplikacija omogućuje korisnicima da jednostavno stvaraju radne naloge s relevantnim informacijama o zadatku, prioritetu, dodijeljenom osoblju, potrebnim resursima i rokovima. Osim toga, MaintainX pruža alate za praćenje napretka radnih naloga, komunikaciju s članovima tima, upravljanje inventarom i dokumentacijom te generiranje izvješća za analizu učinkovitosti održavanja. Ovaj softver je posebno koristan za organizacije koje žele poboljšati produktivnost, efikasnost i transparentnost u svojim procesima održavanja. Aplikacija ima prijavu i registraciju koja joj pomaže pri spremanju korisnikovih radnih naloga.

3. RAZVOJNO OKRUŽENJE

Za izradu ovoga rada korišteno je razvojno okruženje Android Studio za izradu Android aplikacija. Android studio podržava Kotlin i Java programske jezike, a u ovom radu je korišten Kotlin.

3.1 Android Studio

Android Studio, dostupan u [4], je sofisticirano razvojno okruženje koje je namijenjeno za razvoj Android aplikacija. To je službeni alat za razvoj Android softvera od strane Googlea. Kroz jednostavno i intuitivno korisničko sučelje, Android Studio omogućuje programerima da izrađuju visokokvalitetne aplikacije za Android operativni sustav. Podržava Java i Kotlin programske jezike, omogućujući programerima da koriste preferirani jezik za razvoj aplikacija. Jedna od ključnih značajki Android Studija je njegova integracija s drugim alatima i uslugama iz Google ekosustava. To uključuje pristup Google Play uslugama, Firebase platformi i drugim Google API-ima koji olakšavaju implementaciju naprednih funkcionalnosti u aplikacije, kao što su autentifikacija korisnika, upravljanje bazama podataka i analitika. Android Studio pruža snažne alate za izradu korisničkog sučelja aplikacija. Programeri mogu vizualno oblikovati sučelje pomoću grafičkog korisničkog sučelja dizajnera, omogućujući im da brzo i jednostavno postave izgled i raspored elemenata sučelja. Također, Android Studio dolazi s alatima za otklanjanje pogrešaka i testiranje aplikacija. Programeri mogu koristiti emulatore ili stvarne uređaje za testiranje i provjeru performansi aplikacija. Osim toga, podržava integraciju sa sustavom za upravljanje verzijama, poput Git-a, što omogućuje programerima da učinkovito upravljaju verzijama svog koda i surađuju u timskom okruženju. Android Studio redovito dobiva ažuriranja koja donose nove značajke, poboljšanja performansi i ispravke problema unutar programa, kako bi programerima pružio najnovije alate i tehnologije za razvoj aplikacija za Android platformu.

3.2 Programski jezik Kotlin

Kotlin je moderni programski jezik koji je razvijen od strane JetBrainsa. On je dizajniran da bude jednostavan za čitanje i pisanje, pružajući čistu i izražajnu sintaksu. Inter operabilan s Java jezikom, što znači da Kotlin može koristiti Java klase i biblioteke te pozivati Java metode, a isto vrijedi i za Java kod koji može koristiti Kotlin klase i funkcionalnosti. To znači da programeri mogu postupno uvesti Kotlin u postojeće Java projekte, bez potrebe za radikalnim prelaskom na potpuno novi jezik.. Ovaj jezik podržava objektno orijentirano i funkcionalno programiranje, pružajući programerima fleksibilnost u izradi aplikacija. Također ima bogat skup standardnih biblioteka i alata koji olakšavaju razvoj i održavanje aplikacija. Njegova popularnost raste posebno za razvoj Android aplikacija zbog svoje jednostavnosti i podrške za moderne Android platforme. Podržan je od strane Googlea kao službeni jezik za razvoj Android aplikacija, što dodatno potvrđuje njegovu vrijednost i važnost u svijetu programiranja.

4. KORIŠTENE TEHNOLOGIJE

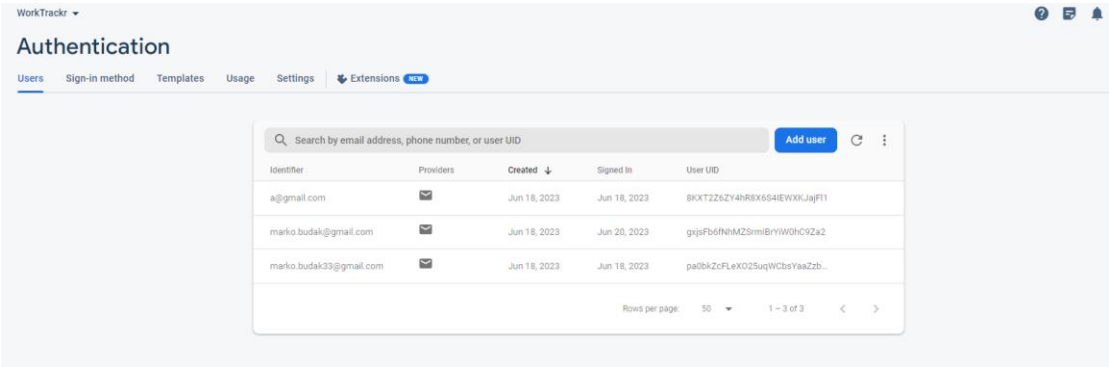
Uz prijašnje navedeni Android Studio i Kotlin programski jezik, za izradu aplikacije se također koristio i Firebase alat.

4.1 Firebase

Firebase, dostupan u [5], je sveobuhvatna platforma u oblaku koja pruža razne usluge i alate za razvoj aplikacija. To je integrirano okruženje koje omogućava programerima da izgrade visokokvalitetne i sigurne aplikacije bez potrebe za infrastrukturom ili složenim postavljanjem poslužitelja. Firebase podržava različite platforme, uključujući mobilne uređaje (Android i iOS), web i desktop. Firebase nudi bazu podataka koja omogućava istovremeno ažuriranje i sinkronizaciju podataka između klijenata i poslužitelja u stvarnom vremenu. To olakšava razvoj aplikacija koje zahtijevaju brzu razmjenu podataka među korisnicima. Autentifikacija korisnika: Firebase ima ugrađenu podršku za autentifikaciju korisnika, što olakšava provjeru identiteta korisnika i upravljanje pristupom aplikacijama. Omogućava prijavu putem e-pošte i lozinke, društvenih mreža, kao i prilagođenu autentifikaciju. Pohrana datoteka: Firebase pruža uslugu pohrane datoteka u oblaku, što omogućava programerima da sigurno pohranjuju i upravljaju različitim vrstama datoteka, poput slika, videozapisa, dokumenta itd. Cloud Messaging: Firebase omogućava programerima slanje *push* obavijesti korisnicima aplikacija na različitim platformama. To je korisno za obavještavanje korisnika o ažuriranjima, važnim obavijestima i interakciji s korisnicima. Analitika: Firebase pruža alate za praćenje korištenja aplikacija, performansi i ponašanja korisnika. Ovi alati omogućavaju programerima da prikupe podatke o aktivnostima korisnika, broju instalacija, zadržavanju korisnika i drugim metrikama kako bi bolje razumjeli svoje korisnike i optimizirali svoje aplikacije. Hosting: Firebase omogućava programerima jednostavne i brze usluge poslužitelja web stranica i web aplikacija. Uz Firebase Hosting, programeri mogu jednostavno objaviti svoje web stranice i aplikacije s visokom dostupnošću i brzim učitavanjem. Machine Learning: Firebase pruža alate za integraciju strojnog učenja u aplikacije. To uključuje vizualno prepoznavanje, strojno prevođenje i druge funkcionalnosti koje koriste napredne tehnike obrade podataka i strojnog učenja.

4.1.2 Firebase autentifikacija korisnika

U ovome radu, za prijavu i registraciju korištena je Firebase autentifikacija korisnika. Provjera autentičnosti korisnika može se provesti pomoću lozinki, telefonskih brojeva ili preko društvenih mreža poput Facebook-a i Twittera. Korištene tehnologije u ovome radu su prijava i registracija pomoću elektroničke pošte i lozinke, te opcija resetiranja lozinke u slučaju da ju je korisnik zaboravio. Primjer registriranih korisnika prikazan je na slici 4.1.

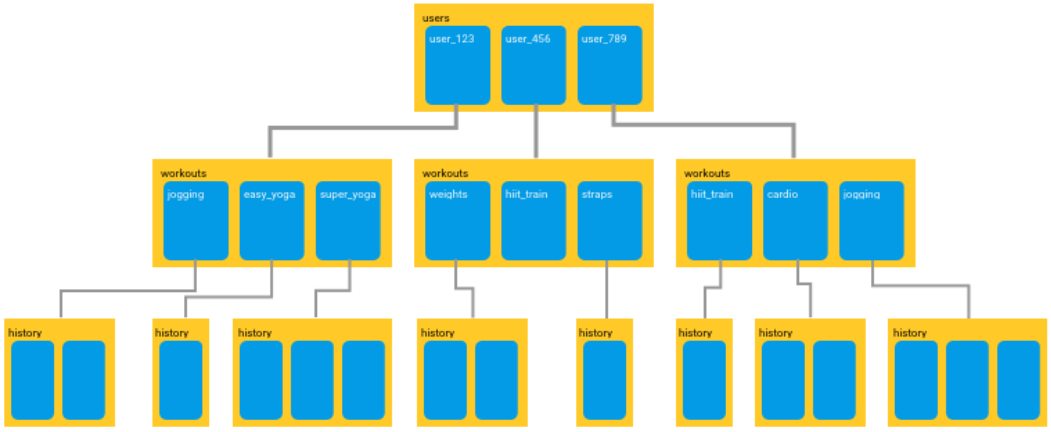


Sl. 4.1. Primjer s već registriranim korisnicima

4.1.3 Firebase Firestore

Za pohranu podataka u bazu, Firebase pruža dvije opcije: Firestore Databse i Realtime Database. Firebase Realtime Database je usluga baze podataka u oblaku koja omogućava programerima da pohranjuju i sinkroniziraju podatke u stvarnom vremenu. To je fleksibilna JSON baza podataka koja omogućava aplikacijama da brzo dijele podatke između klijentskih uređaja i poslužitelja. Firestore, dostupan u [6], je još jedna usluga baze podataka u oblaku koja je također dio Firebase platforme. Firestore je dokument-orijentirana baza podataka koja pruža naprednije funkcionalnosti u odnosu na Firebase Realtime Database. Pohranjuje podatke u obliku dokumenata koji se organiziraju u kolekcije. Svaki dokument je mapa koja sadrži ključ-vrijednost parove podataka. Ova struktura olakšava organizaciju i upravljanje podacima u hijerarhijskom obliku. Slično kao Firebase Realtime Database, Firestore podržava sinkronizaciju podataka u stvarnom vremenu, što znači da se promjene podataka automatski ažuriraju na svim povezanim uređajima. U ovome radu korišten je Firebase Firestore Database

zbog lakše upotrebe i boljih funkcionalnosti. Struktura Firestore baze podataka je prikazana na slici 4.2.



Slika 4.2. Primjer strukture Firestore baze podataka

5. PROGRAMSKO RJEŠENJE ZA IZRADU APLIKACIJE

Za uspješno i kvalitetno kreiranje aplikacije prolazi se kroz službenu dokumentaciju te se koristi internetskim stranicama poput GeeksForGeeks i W3Schools za upoznavanje sa svim potrebnim tehnologijama.

5.1. Početni zaslon

Pri prvom ulasku u aplikaciju, korisnik ima mogućnost registracije ili, ako je već postojeći korisnik, prijave u aplikaciju. Pomoću *binding* varijable, omogućava se navigacija između više aktivnosti unutar aplikacije. Također se poziva i autentifikacija pomoću Firebase-a. Ako je korisnik ostao prijavljen, aplikacija ga direktno vodi na aktivnost za kreiranje radnoga naloga. Kod glavne aktivnosti je prikazan na slici 5.1.

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var auth : FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        supportActionBar?.hide()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //Autentifikacija
        auth = FirebaseAuth.getInstance()

        binding.btnLogin.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, LoginActivity::class.java))
        }
        binding.btnRegister.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, RegisterActivity::class.java))
        }
    }

    override fun onStart() {
        super.onStart()
        if(auth.currentUser != null){
            Intent( packageContext: this, HomeActivity::class.java).also { it: Intent
                it.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                startActivity(it)
            }
        }
    }
}
```

Slika 5.1. Kod glavne aktivnosti

5.2. Registracija korisnika

U aktivnosti za registraciju, korisnik upisuje svoju mail adresu, ime i prezime, naziv firme, lozinku i konfirmaciju lozinke. Stavljeni su uvjeti za neke od vrijednosti kako korisnik ne bi ostavio važne podatke prazne ili stavio krivi format mail adrese. Kod za unos podataka prikazan je slikom 5.2.

```
class RegisterActivity : AppCompatActivity() {
    private lateinit var binding: ActivityRegisterBinding
    private lateinit var auth : FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        supportActionBar?.hide()
        binding = ActivityRegisterBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //Autentifikacija
        auth = FirebaseAuth.getInstance()

        val nameCheck = RxTextView.textChanges(binding.fullname) InitialValueObservable<CharSequence!>
            .skipInitialValue() Observable<CharSequence!>!
            .map { name -> name.isEmpty() }
        nameCheck.subscribe { showNameExistsAlert(it) }

        val emailCheck = RxTextView.textChanges(binding.etEmail) InitialValueObservable<CharSequence!>
            .skipInitialValue() Observable<CharSequence!>!
            .map { email -> !Patterns.EMAIL_ADDRESS.matcher(email).matches() }
        emailCheck.subscribe { showEmailValidAlert(it) }

        val passCheck = RxTextView.textChanges(binding.etPassword) InitialValueObservable<CharSequence!>
            .skipInitialValue() Observable<CharSequence!>!
            .map { pass -> pass.length < 8 }
        passCheck.subscribe { showTextMinimalAlert(it, text: "Password") }

        val passConfirmCheck = io.reactivex.Observable.merge(
            RxTextView.textChanges(binding.etPassword) InitialValueObservable<CharSequence!>
                .skipInitialValue() Observable<CharSequence!>!
                .map { pass -> pass.toString() != binding.etConfirmPassword.text.toString()
                },
            RxTextView.textChanges(binding.etConfirmPassword) InitialValueObservable<CharSequence!>
                .skipInitialValue() Observable<CharSequence!>!
                .map { confirmPass -> confirmPass.toString() != binding.etPassword.text.toString() })
        passConfirmCheck.subscribe { it: Boolean!
            showPasswordConfirmAlert(it)
        }
```

Slika 5.2. Unos podataka za registraciju

Gumb za registraciju je siv i ne može se kliknuti sve dok nisu zadovoljeni svi uvjeti unosa podataka pri registraciji. Nakon unosa svih podataka gumb postaje zelen i može se kliknuti. Klik na gumb korisnika vodi na glavnu aktivnost gdje može započeti sa kreiranjem radnoga naloga. U slučaju da korisnik već registriran, postoji mogućnost direktnog odlaska na aktivnost za prijavu. Kod za vrednovanje unesenih podataka prikazan je slikom 5.3.

```
        val invalidFields = io.reactivex.Observable.combineLatest(
            nameCheck,
            emailCheck,
            passCheck,
            passConfirmCheck
        ) { nameInvalid: Boolean, emailInvalid: Boolean, passInvalid: Boolean, passConfirmInvalid: Boolean ->
            !nameInvalid && !emailInvalid && !passInvalid && !passConfirmInvalid
        }
        invalidFields.subscribe { isValid ->
            if(isValid) {
                binding.btnRegister.isEnabled = true
                binding.btnRegister.backgroundTintList = ContextCompat.getColorStateList( context: this, R.color.primary_color)
            } else {
                binding.btnRegister.isEnabled = false
                binding.btnRegister.backgroundTintList = ContextCompat.getColorStateList( context: this, android.R.color.darker_gray)
            }
        }

        binding.btnRegister.setOnClickListener { it: View!
            val email = binding.etEmail.text.toString().trim()
            val password = binding.etPassword.text.toString().trim()
            val fullName = binding.fullname.text.toString().trim()
            val firmName = binding.etFirmName.text.toString().trim()
            registerUser(email, password, fullName, firmName )
        }
        binding.tvHasAccount.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, LoginActivity::class.java))
        }
    }
```

Slika 5.3. Provjera unosa podataka i aktiviranje gumba za registraciju

U funkciji *registerUser*, pomoću *SharedPreferences*, dostupan na [7], aplikacija nakon registracije dobiva ID korisnika te prema tom ID-u u kolekcije sprema korisnikovo puno ime i naziv firme za koju radi. Kod funkcije za registraciju prikazan je slikom 5.4.

```
    private fun registerUser(email: String, password: String, fullName: String, firmName: String) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    val user = auth.currentUser
                    val userId = user?.uid

                    val sharedPref = getSharedPreferences( name: "WorkTrackr", Context.MODE_PRIVATE)
                    val editor = sharedPref.edit()
                    editor.putString("userId", userId)
                    editor.apply()

                    val db = FirebaseFirestore.getInstance()
                    val fullNameData = hashMapOf(
                        "fullName" to fullName
                    )
                    val firmNameData = hashMapOf(
                        "firmName" to firmName
                    )
                    if (userId != null) {
                        db.collection( collectionPath: "fullName").document(userId)
                            .set(fullNameData)
                        db.collection( collectionPath: "firmName").document(userId) DocumentReference
                            .set(firmNameData) Task<Void!>

                            .addOnSuccessListener { it: Void!
                                startActivity(Intent( packageContext: this, LoginActivity::class.java))
                                Toast.makeText( context: this,  text: "Registration successful!", Toast.LENGTH_SHORT).show()
                            }
                            .addOnFailureListener { exception ->
                                Toast.makeText( context: this,  text: "Registration successful, but failed to save data: $exception", Toast.LENGTH_SHORT).show()
                            }
                    }
                } else {
                    Toast.makeText( context: this, task.exception?.message, Toast.LENGTH_SHORT).show()
                }

            }
    }
```

Slika 5.4. Funkcija za registraciju

5.3. Prijava korisnika

Nakon registracije, aplikacija korisnika dovodi do aktivnosti za prijavu. Korisnik se prijavljuje s mail adresom i lozinkom koji je unio pri registraciji. Postoje uvjeti da mail adresa i lozinka ne smiju biti prazan tekst. Ukoliko korisnik nije registriran, omogućen je direktan odlazak na aktivnost za registraciju. Kod s uvjetima za prijavu prikazan je slikom 5.5.

```
        binding.btnLogin.setOnClickListener { it: View!
            val email = binding.email.text.toString().trim()
            val password = binding.password.text.toString().trim()
            loginUser(email, password)
        }
        binding.tvNoAccount.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, RegisterActivity::class.java))
        }

        binding.tvForgotPassword.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, ResetPassActivity::class.java))
        }
    }

    private fun showTextMinimalAlert(isNotValid: Boolean, text : String) {
        if (text == "Email")
            binding.email.error = if(isNotValid) "$text Can not be empty!" else null
       else if (text == "Password")
            binding.password.error = if(isNotValid) "$text Can not be empty!" else null

    }
```

Slika 5.5. Uvjeti pri prijavi

Funkcija za prijavu *loginUser* pomoću *SharedPreferences* dohvaća ID korisnika. Ukoliko ID korisnika nije dodan i pri prijavi, nakon registracije novog korisnika radni nalozi se spremaju u kolekciju krivoga korisnika. Kod funkcije za prijavu prikazan je slikom 5.6.

```
    private fun loginUser(email: String, password: String) {
        auth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { login ->
                if (login.isSuccessful) {
                    val user = auth.currentUser
                    val userId = user?.uid

                    val sharedPref = getSharedPreferences( name: "WorkTrackr", Context.MODE_PRIVATE)
                    val editor = sharedPref.edit()
                    editor.putString("userId", userId)
                    editor.apply()

                    Intent( packageContext: this, HomeActivity::class.java).also { it: Intent
                        it.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                        startActivity(it)
                        Toast.makeText( context: this, text: "Login successful!", Toast.LENGTH_SHORT).show()
                    }
                } else {
                    Toast.makeText( context: this, login.exception?.message, Toast.LENGTH_SHORT).show()
                }
    }}
```

Slika 5.6. Funkcija za prijavu

5.4. Resetiranje lozinke

U slučaju da je korisnik zaboravio lozinku za prijavu, moguće je resetirati lozinku. Korisnik upisuje svoju email adresu te se na istu šalje link za resetiranje lozinke. Pomoću funkcije *showEmailValidAlert* napravljen je uvjet da format email adrese mora biti točan. Kod aktivnosti za resetiranje lozinke prikazan je slikom 5.7.

```
class ResetPassActivity : AppCompatActivity() {

    private lateinit var binding: ActivityResetPassBinding
    private lateinit var auth : FirebaseAuth

    @SuppressLint("CheckResult")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        supportActionBar?.hide()
        binding = ActivityResetPassBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //Autentifikacija
        auth = FirebaseAuth.getInstance()

        val emailCheck = RxTextView.textChanges(binding.email) InitialValueObservable<CharSequence!>
            .skipInitialValue() Observable<CharSequence!>!
            .map { email -> !Patterns.EMAIL_ADDRESS.matcher(email).matches() }
        emailCheck.subscribe { showEmailValidAlert(it) }

        binding.btnReset.setOnClickListener {  it: View!
            val email = binding.email.text.toString().trim()
            auth.sendPasswordResetEmail(email)
                .addOnCompleteListener(this){reset->
                    if(reset.isSuccessful){
                        Intent( packageContext: this, LoginActivity::class.java).also {  it: Intent
                            it.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                            startActivity(it)
                            Toast.makeText( context: this,  text: "Check your email for password reset!", Toast.LENGTH_SHORT).show()
                        }
                    }else{
                        Toast.makeText( context: this, reset.exception?.message ,Toast.LENGTH_SHORT).show()
                    }
                }
        }

        binding.tvBackLogin.setOnClickListener {  it: View!
            startActivity(Intent( packageContext: this, LoginActivity::class.java))
        }
    }

    private fun showEmailValidAlert(isNotValid: Boolean) {
        if(isNotValid){
        binding.email.error =  "Invalid email!"
        binding.btnReset.isEnabled = false
        binding.btnReset.backgroundTintList = ContextCompat.getColorStateList( context: this, android.R.color.darker_gray)
    } else {
        binding.email.error = null
            binding.btnReset.isEnabled = true
            binding.btnReset.backgroundTintList = ContextCompat.getColorStateList( context: this, R.color.primary_color)
        }
    }
}
```

Slika 5.7. Aktivnost za resetiranje lozinke

5.5. Kreiranje radnoga naloga

Nakon uspješne prijave, korisnika se dovodi do aktivnosti za kreiranje radnoga naloga. Korisnik ima opciju odabira vrste posla, pod posla te upisivanje materijala i alata koji su korišteni. Odabir vrste posla napravljen je pomoću kliznog izbornika. Poslovi koju se mogu odabrati su spremljeni u kolekciju unutar Firestore baze podataka. Kod kliznog izbornika za odabir vrste posla prikazan je slikom 5.8.

```
val jobOptions: MutableList<String> = mutableListOf()
val mainJobsCollectionRef = FirebaseFirestore.getInstance().collection( collectionPath: "mainJobs").orderBy( field: "name", Query.Direction.ASCENDING)

mainJobsCollectionRef.get()
    .addOnSuccessListener { snapshot ->
        for (document in snapshot.documents) {
            val jobName = document.getString( field: "name")
            jobName?.let { jobOptions.add(it) }
        }
        jobOptions.add( index: 0,  element: "Select job type")
        val adapter = object : ArrayAdapter<String>(
             context: this,
            android.R.layout.simple_spinner_item,
            jobOptions
        ) {
            override fun getDropDownView(position: Int, convertView: View?, parent: ViewGroup): View {
                val view = super.getDropDownView(position, convertView, parent)
                view.setBackgroundColor(Color.WHITE)

                val textView = view.findViewById<TextView>(android.R.id.text1)
                textView.setTextColor(Color.DKGRAY)

                return view
            }

            override fun isEnabled(position: Int): Boolean {
                return position != 0
            }
            override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
                val view = super.getView(position, convertView, parent)
                if (position == 0) {
                    (view as TextView).setTextColor(Color.GRAY)
                } else {
                    (view as TextView).setTextColor(Color.BLACK)
                }
                return view
            }
        }
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        spinner.adapter = adapter
    }
    .addOnFailureListener { exception ->
        Log.e( tag: "Spinner",  msg: "Error retrieving job options: $exception")
    }
spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
        val selectedOption = parent?.getItemAtPosition(position).toString()
        textView.text = selectedOption

    }
    override fun onNothingSelected(parent: AdapterView<*>?) {

    }

}
```

Slika 5.8. Klizni izbornik za vrstu posla

Odabir pod posla uvjetovan je odabirom vrste posla. Klikom na klizni izbornik prikazuju se samo pod poslovi koji pripadaju odabranoj vrsti posla. Pod poslovi se nalaze u Firestore kolekcijama. U slučaju da korisnik nije odabrao niti jednu vrstu

posla, u drugome kliznome izborniku se prikazuje samo tekst *Select your job*. Primjer jednog od kliznih izbornika pod poslova prikazan je slikom 5.9.

```
spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
        val selectedOption = parent?.getItemAtPosition(position).toString()
        textView.text = selectedOption

        when (selectedOption) {
            "Law" -> {
                val lawJobsCollectionRef = FirebaseFirestore.getInstance().collection( collectionPath: "law").orderBy( field: "nameLaw", Query.Direction.ASCENDING)

                lawJobsCollectionRef.get()
                    .addOnSuccessListener { snapshot ->
                        val jobLawOptions: MutableList<String> = mutableListOf()

                        for (document in snapshot.documents) {
                            val jobNameLaw = document.getString( field: "nameLaw")
                            jobNameLaw?.let { jobLawOptions.add(it) }
                        }
                        jobLawOptions.add( index: 0,  element: "Select your job")
                        val adapter = object : ArrayAdapter<String>(
                            applicationContext,
                            android.R.layout.simple_spinner_item,
                            jobLawOptions
                        ) {
                            override fun getDropDownView(position: Int, convertView: View?, parent: ViewGroup): View {
                                val view = super.getDropDownView(position, convertView, parent)
                                view.setBackgroundColor(Color.WHITE)

                                val textView = view.findViewById<TextView>(android.R.id.text1)
                                textView.setTextColor(Color.DKGRAY)
                                return view
                            }

                            override fun isEnabled(position: Int): Boolean {
                                return position != 0
                            }

                            override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
                                val view = super.getView(position, convertView, parent)
                                if (position == 0) {
                                    (view as TextView).setTextColor(Color.GRAY)
                                } else {
                                    (view as TextView).setTextColor(Color.BLACK)
                                }
                                return view
                            }
                        }
                        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
                        spinnerLaw.adapter = adapter
                        spinnerLaw.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
                            override fun onItemSelected(parent: AdapterView<*>?, view: View?, position: Int, id: Long) {
                                val selectedOption = parent?.getItemAtPosition(position).toString()
                                textViewLaw.text = selectedOption
                            }

                            override fun onNothingSelected(parent: AdapterView<*>?) {
                            }
                        }
                    }
```

Slika 5.9. Klizni izbornik pod poslova za *Law* vrstu posla

Nakon izbora poslova te popunjavanja polja za korištene materijale i alate po potrebi, korisnik može započeti mjerenje rada klikom na gumb. Pritiskom na gumb *punchIn* navedeni gumb nestaje te se prikazuju gumbovi za pauziranje rada i završetak rada. Dodana je varijabla *punchInTime* koja prikazuje datum i vrijeme pri pritisku gumba za početak rada. Varijabla *elapsedTime* prikazuje vrijeme koje korisnik provede radeći. Funkcija za početak rada štoperica prikazana je slikom 5.10.

```
private fun startTimer() {
    if (!isTimerRunning) {
        isTimerRunning = true

        punchInTime = System.currentTimeMillis()
        val format = SimpleDateFormat( pattern: "dd/MM/yyyy; hh:mm:ss a")
        punchInTimeFormatted = format.format(Date(punchInTime))

        startTime = System.currentTimeMillis() - pausedTime
        btnPunchIn.visibility = View.GONE
        btnPause.visibility = View.VISIBLE
        tvPausedTime.visibility = View.VISIBLE
        tvElapsedTime.visibility = View.VISIBLE
        btnPause.text = "Pause"
        btnPunchOut.visibility = View.VISIBLE

        timer = object : CountDownTimer(Long.MAX_VALUE, countDownInterval: 1000) {
            override fun onTick(millisUntilFinished: Long) {
                val currentTime = System.currentTimeMillis()
                val elapsedTime = currentTime - startTime
                updateElapsedTime(elapsedTime)
            }
            override fun onFinish() {
            }
        }.start()
    } else {
        resumeTimer()
    }
}
```

Slika 5.10. Funkcija za početak rada štoperice

Korisnik ima mogućnost pauziranja rada i odlaska na pauzu klikom na *btnPause* gumb. Mjerenje vremena rada se pauzira te se mjerenje vremena provedenog na pauzi počinje mjeriti. Ponovnim pritiskom na gumb prekida se pauza te se nastavlja s mjerenjem vremena provedenog radeći. Kod funkcije za odlazak na pauzu prikazan je slikom 5.11., a slikom 5.12. je prikazan kod funkcije za prekidanje pauze i nastavka s radom.

```
private fun pauseTimer() {
    if (isTimerRunning) {
        timer?.cancel()
        isTimerRunning = false
        btnPause.text = "Continue"

        val currentTime = System.currentTimeMillis()
        pausedTimeStart = currentTime // zapocni mjeriti vrijeme

        pausedTimeTimer?.cancel()

        pausedTimeTimer = object : CountDownTimer(Long.MAX_VALUE, countDownInterval: 1000) {
            override fun onTick(millisUntilFinished: Long) {
                val currentTimer = System.currentTimeMillis()
                val currentPausedTime = currentTimer - pausedTimeStart + pausedTime
                updatePausedTime(currentPausedTime) // Update the paused time display
            }

            override fun onFinish() {
            }
        }.start()
    }
}
```

Slika 5.11. Funkcija za odlazak na pauzu

```
private fun resumeTimer() {
    if (!isTimerRunning) {
        isTimerRunning = true
        btnPause.text = "Pause"

        pausedTimeTimer?.cancel()

        val currentTime = System.currentTimeMillis()
        pausedTime += currentTime - pausedTimeStart

        timer = object : CountDownTimer(Long.MAX_VALUE, countDownInterval: 1000) {
            override fun onTick(millisUntilFinished: Long) {
                val currentTimer = System.currentTimeMillis()
                val elapsedTime = currentTimer - startTime - pausedTime
                updateElapsedTime(elapsedTime)
            }

            override fun onFinish() {
            }
        }.start()
    }
}
```

Slika 5.12. Funkcija za prekidanje pauze i nastavka s radom

Nakon što je korisnik gotov s radom, klikom na gumb *bntPunchOut* završava s radom i prelazi na aktivnost za prikaz podataka o radu. Varijabla *punchOutTime* prikazuje datum i vrijeme kada je korisnik pritisnuo gumb za završetak rada. Pomoću *intent*, dostupan na [8], svi podatci koje je korisnik odabrao, unio i koristio za kreiranje radnoga naloga se šalju na aktivnost za prikaz podataka o radu. Kod funkcije za završetak rada prikazan je slikom 5.13.

```
private fun stopTimer() {
    timer?.cancel()
    isTimerRunning = false
    punchOutTime = System.currentTimeMillis()
    val format = SimpleDateFormat( pattern: "dd/MM/yyyy; hh:mm:ss a")
    val punchOutTimeFormatted = format.format(Date(punchOutTime))
    //vremena i poslovi
    val intent = Intent( packageContext: this, WorkInfoActivity::class.java)
    intent.putExtra( name: "startTime",  value: "Start time: $punchInTimeFormatted")
    intent.putExtra( name: "endTime",  value: "End time: $punchOutTimeFormatted")
    intent.putExtra( name: "elapsedTime", tvElapsedTime.text.toString())
    intent.putExtra( name: "pausedTime", tvPausedTime.text.toString())
    intent.putExtra( name: "selectedOption",  value: "Branch of job: " + textView.text.toString())
    intent.putExtra( name: "selectedJob",  value: "Your job: " + textViewLaw.text.toString())
    //Materijali
    val materialsUsedText = findViewById<TextInputEditText>(R.id.materials_used_text)
    val textMaterial = materialsUsedText.text.toString()
    DataHolder.materialsUsed = "Materials used: $textMaterial"
    //Toolovi
    val toolsUsedText = findViewById<TextInputEditText>(R.id.tools_used_text)
    val textTool = toolsUsedText.text.toString()
    DataToolsHolder.toolsUsed = "Tools used: $textTool"
    startActivity(intent)

    startTime = 0
    pausedTime = 0
    pausedTimeStart = 0

    pausedTimeTimer?.cancel()
    pausedTimeTimer = null
    pausedTimeStart = System.currentTimeMillis()
    updateElapsedTime( elapsedTime: 0)
    updatePausedTime( pausedTime: 0)

    btnPunchIn.visibility = View.VISIBLE
    btnPause.visibility = View.GONE
    btnPunchOut.visibility = View.GONE
    tvPausedTime.visibility = View.GONE
    tvElapsedTime.visibility = View.GONE
}
```

Slika 5.13. Funkcija za završetak rada

Dodatne mogućnosti aktivnosti su *btnLogout* koji korisniku daje opciju odjave iz aplikacije. Dodan je i *btnCollection* koji korisnika odvodi na aktivnost gdje se prikazuju svi radni nalozi koje je korisnik kreirao. Pomoću *calendarBtn* korisnik može provjeriti datume. Kod za dodatne mogućnosti prikazan je slikom 5.14.

```
binding.btnLogout.setOnClickListener {  it: View!
    auth.signOut()
    Intent( packageContext: this, MainActivity::class.java).also {  it: Intent
        it.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(it)
        Toast.makeText( context: this,  text: "Logout successful!", Toast.LENGTH_SHORT).show()
    }
}

binding.btnCollection.setOnClickListener {  it: View!
    startActivity(Intent( packageContext: this, CollectionActivity::class.java))
}

val calendar = Calendar.getInstance()
val day = calendar.get(Calendar.DAY_OF_MONTH)
val month = calendar.get(Calendar.MONTH)
val year = calendar.get(Calendar.YEAR)


binding.calendarBtn.setOnClickListener {  it: View!
    val dpd = DatePickerDialog( context: this, DatePickerDialog.OnDateSetListener { _, _, _, _ ->

    }, year, month, day)
    dpd.show()
}
```

Slika 5.14. Dodatne mogućnosti aktivnosti

5.6. Spremanje ili brisanje radnoga naloga

Nakon završetka rada, korisnika se dovodi na *WorkInfoActivity*. Ovdje su korisniku prikazani svi podatci koje je unio ili koristio pri radu. Podatci su preneseni preko *intent* a korisnički ID je dohvaćen preko prijašnje spomenutog *SharedPreferences*. Pritiskom na *homeBtn* korisnik briše radni nalog te se vraća na aktivnost za kreiranje radnoga naloga. Kod dohvaćanja podataka s *intent* prikazan je na slici 5.15.

```
binding.homeBtn.setOnClickListener {  it: View!
    startActivity(Intent( packageContext: this, HomeActivity::class.java))
    Toast.makeText( context: this,  text: "Work order deleted!", Toast.LENGTH_SHORT).show()
}


val startTime = intent.getStringExtra( name: "startTime")
val endTime = intent.getStringExtra( name: "endTime")
val elapsedTime = intent.getStringExtra( name: "elapsedTime")
val pausedTime = intent.getStringExtra( name: "pausedTime")

val tvStartTime = findViewById<TextView>(R.id.pocetak)
val tvEndTime = findViewById<TextView>(R.id.kraj)
val tvElapsedTime = findViewById<TextView>(R.id.elaps)
val tvPausedTime = findViewById<TextView>(R.id.pauza)

tvStartTime.text = startTime
tvEndTime.text = endTime
tvElapsedTime.text = elapsedTime
tvPausedTime.text = pausedTime

val sharedPref = getSharedPreferences( name: "WorkTrackr", Context.MODE_PRIVATE)
val userId = sharedPref.getString( key: "userId",  defValue: "")

if (userId != null) {
    retrieveFullName(userId)
    retrieveFirmName(userId)
}
val materialsUsed = DataHolder.materialsUsed
val materialTextView = findViewById<TextView>(R.id.material)
materialTextView.text = materialsUsed

val toolsUsed = DataToolsHolder.toolsUsed
val toolTextView = findViewById<TextView>(R.id.tool)
toolTextView.text = toolsUsed

val selectedOption = intent.getStringExtra( name: "selectedOption")
val optionSelected = findViewById<TextView>(R.id.jobType)
optionSelected.text = selectedOption

val selectedJob = intent.getStringExtra( name: "selectedJob")
val jobSelected = findViewById<TextView>(R.id.job)
jobSelected.text = selectedJob
```

Slika 5.15. Dohvaćanje podataka pomoću *intent*

Problema dohvaćanja punog imena i naziva firme korisnika se rješava kreiranjem funkcija za dohvaćanja navedenih varijabli preko traženih kolekcija pomoću korisničkog ID-a. Varijable *nameFull* i *nameFirm* se pune s dohvaćenim podatcima iz kolekcija da bi se moglo u sljedećim koracima prikazati te podatke. Kod funkcije za dohvaćanje punog imena i naziva firme korisnika prikazan je slikom 5.16.

```
    @SuppressLint("SetTextI18n")
    private fun retrieveFullName(userId: String) {
        val fullNameRef = FirebaseFirestore.getInstance().collection( collectionPath: "fullName").document(userId)

        fullNameRef.get()
            .addOnSuccessListener { documentSnapshot ->
                val fullName = documentSnapshot.getString( field: "fullName")
                val nameTextView = findViewById<TextView>(R.id.name)
                nameTextView.text = "Full name: $fullName"
                if (fullName != null) {
                    nameFull = fullName
                }
            }
            .addOnFailureListener { exception ->
                Log.e( tag: "firestore",  msg: "Error retrieving: $exception")
            }
    }

    @SuppressLint("SetTextI18n")
    private fun retrieveFirmName(userId: String) {
        val firmNameRef = FirebaseFirestore.getInstance().collection( collectionPath: "firmName").document(userId)

        firmNameRef.get()
            .addOnSuccessListener { documentSnapshot ->
                val firmName = documentSnapshot.getString( field: "firmName")
                val firmTextView = findViewById<TextView>(R.id.firm)
                firmTextView.text = "Firm name: $firmName"
                if (firmName != null) {
                    nameFirm = firmName
                }
            }
            .addOnFailureListener { exception ->
                Log.e( tag: "firestore",  msg: "Error retrieving: $exception")
            }
    }
}
```

Slika 5.16. Funkcije za dohvaćanje punog imena i naziva firme korisnika

Pritiskom na gumb *addBtn* korisnik sprema svoj radni nalog. Radni nalozi se spremaju u kolekciju *users* te se unutar te kolekcije stvaraju dokumenti pomoću korisničkog ID-a. Unutar tih dokumenata, kreiraju se pod kolekcije u kojima se spremaju i prikazuju radni nalozi korisnika. Kod za spremanje radnih naloga prikazan je slikom 5.17.

```
    binding.addBtn.setOnClickListener { it: View!
        val firestore = FirebaseFirestore.getInstance()
        val usersCollectionRef = firestore.collection( collectionPath: "users")
        val userDocumentRef = if (userId != null) usersCollectionRef.document(userId) else null

        if (userDocumentRef != null) {
            val naloziCollectionRef = userDocumentRef.collection( collectionPath: "nalozi")
            val workEntryDocumentRef = naloziCollectionRef.document()

            val workEntryInfo = hashMapOf(
                "startTime" to startTime,
                "endTime" to endTime,
                "elapsedTime" to elapsedTime,
                "pausedTime" to pausedTime,
                "materialsUsed" to materialsUsed,
                "toolsUsed" to toolsUsed,
                "jobType" to selectedOption,
                "yourJob" to selectedJob,
                "firmName" to nameFirm,
                "fullName" to nameFull
            )

            workEntryDocumentRef.set(workEntryInfo)
                .addOnSuccessListener { it: Void!
                }
                .addOnFailureListener { exception ->
                    Log.e( tag: "firestore",  msg: "Error adding document: $exception")
                }

            naloziCollectionRef.get()
                .addOnSuccessListener { querySnapshot ->
                    for (documentSnapshot in querySnapshot.documents) {
                        documentSnapshot.getString( field: "startTime")
                        documentSnapshot.getString( field: "endTime")
                        documentSnapshot.getString( field: "elapsedTime")
                        documentSnapshot.getString( field: "pausedTime")
                        documentSnapshot.getString( field: "materialsUsed")
                        documentSnapshot.getString( field: "toolsUsed")
                        documentSnapshot.getString( field: "jobType")
                        documentSnapshot.getString( field: "yourJob")
                        documentSnapshot.getString( field: "firmName")
                        documentSnapshot.getString( field: "fullName")
                    }
                    Toast.makeText( context: this,  text: "Work order saved!", Toast.LENGTH_SHORT).show()
                }
                .addOnFailureListener { exception ->
                    Log.e( tag: "firestore",  msg: "Error retrieving documents: $exception")
                }
        }
        startActivity(Intent( packageContext: this, HomeActivity::class.java))
    }
}
```

Slika 5.17. Spremanje radnih naloga

5.7. Kolekcija radnih naloga

Iz aktivnosti gdje se kreira radni nalog, korisnik može klikom na gumb preći na aktivnost koja prikazuje kolekciju svih korisnikovih radnih naloga. Podatci se dohvaćaju iz Firestore kolekcije, a prikazuju se pomoću *RecyclerView*, dostupan na [9], i njegove adapter klase, dostupan na[10]. Kod adaptera prikazan je slikom 5.18., a slikom 5.19. prikazan je kod za dohvaćanje radnih naloga korisnika

```
class CollectionAdapter(private val items: List<CollectionItem>) : RecyclerView.Adapter<CollectionAdapter.CollectionViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CollectionViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.collection_recycler_view, parent, attachToRoot: false)
        return CollectionViewHolder(view)
    }

    override fun onBindViewHolder(holder: CollectionViewHolder, position: Int) {

        when (holder) {
            is CollectionViewHolder -> {
                holder.bind(position, items[position])
            }
        }
    }

    override fun getItemCount(): Int {
        return items.size
    }

    class CollectionViewHolder(view: View) : RecyclerView.ViewHolder(view) {

        val startTime: TextView = view.findViewById(R.id.pocetakRecycler)
        val endTime: TextView = view.findViewById(R.id.krajRecycler)
        val elapsedTime: TextView = view.findViewById(R.id.elapsRecycler)
        val pausedTime : TextView = view.findViewById(R.id.pauzaRecycler)
        val materialsUsed : TextView = view.findViewById(R.id.materialRecycler)
        val toolsUsed : TextView = view.findViewById(R.id.toolRecycler)
        val jobType : TextView = view.findViewById(R.id.jobTypeRecycler)
        val yourJob : TextView = view.findViewById(R.id.jobRecycler)
        val firmName : TextView = view.findViewById(R.id.firmRecycler)
        val fullName : TextView = view.findViewById(R.id.nameRecycler)

        @SuppressLint("SetTextI18n")
        fun bind(
            index : Int,
            collectionItem: CollectionItem
        ){
            startTime.text = collectionItem.startTime
            endTime.text = collectionItem.endTime
            elapsedTime.text = collectionItem.elapsedTime
            pausedTime.text = collectionItem.pausedTime
            materialsUsed.text = collectionItem.materialsUsed
            toolsUsed.text = collectionItem.toolsUsed
            jobType.text = collectionItem.jobType
            yourJob.text = collectionItem.yourJob
            firmName.text = "Firm name: " + collectionItem.firmName
            fullName.text = "Full name: " + collectionItem.fullName
        }
    }

}
```

Slika 5.18. CollectionAdapter kod

```
        binding.toHomeBtn.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, HomeActivity::class.java))
        }

        recyclerViewCollection = findViewById(R.id.recyclerViewCollection)

        val sharedPref = getSharedPreferences( name: "WorkTrackr", Context.MODE_PRIVATE)
        val userID = sharedPref.getString( key: "userId",  defValue: "")

        if (userID != null) {
            retrieveDocumentsFromCollection(userID)
        }
    }

    @SuppressLint("NotifyDataSetChanged")
    private fun retrieveDocumentsFromCollection(userID: String) {
        val collectionRef = FirebaseFirestore.getInstance().collection( collectionPath: "users").document(userID).collection( collectionPath: "nalozi").orderBy( field: "startTime", Query.Direction.ASCENDING)
        collectionRef.get()
            .addOnSuccessListener { querySnapshot ->
                for (documentSnapshot in querySnapshot) {
                    val startTime = documentSnapshot.getString( field: "startTime")
                    val endTime = documentSnapshot.getString( field: "endTime")
                    val elapsedTime = documentSnapshot.getString( field: "elapsedTime")
                    val pausedTime = documentSnapshot.getString( field: "pausedTime")
                    val materialsUsed = documentSnapshot.getString( field: "materialsUsed")
                    val toolsUsed = documentSnapshot.getString( field: "toolsUsed")
                    val jobType = documentSnapshot.getString( field: "jobType")
                    val yourJob = documentSnapshot.getString( field: "yourJob")
                    val firmName = documentSnapshot.getString( field: "firmName")
                    val fullName = documentSnapshot.getString( field: "fullName")

                    val item = CollectionItem(
                        startTime.toString(), endTime.toString(), elapsedTime.toString(),
                        pausedTime.toString(), materialsUsed.toString(), toolsUsed.toString(),
                        jobType.toString(), yourJob.toString(), firmName.toString(), fullName.toString()
                    )
                    collectionItems.add(item)
                }
                adapter = CollectionAdapter(collectionItems)
                recyclerViewCollection.adapter = adapter
                recyclerViewCollection.apply { this: RecyclerView
                    layoutManager = LinearLayoutManager( context: this@CollectionActivity)

                }

                adapter.notifyDataSetChanged()
            }
            .addOnFailureListener { exception ->
                Log.e( tag: "Firestore",  msg: "Error retrieving documents: $exception")
            }
    }
}
```

Slika 5.19. Dohvaćanje radnih naloga korisnika

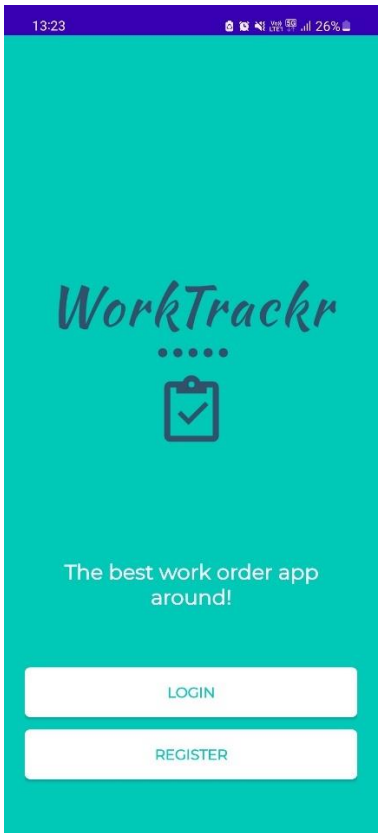
6. IZGLED APLIKACIJE

Mobilna aplikacija se sastoji od sedam zaslona:

- Početni zaslon
- Registracija korisnika
- Prijava korisnika
- Resetiranje lozinke
- Kreiranje radnoga naloga
- Spremanje ili brisanje radnoga naloga
- Kolekcija radnih naloga

6.1. Početni zaslon

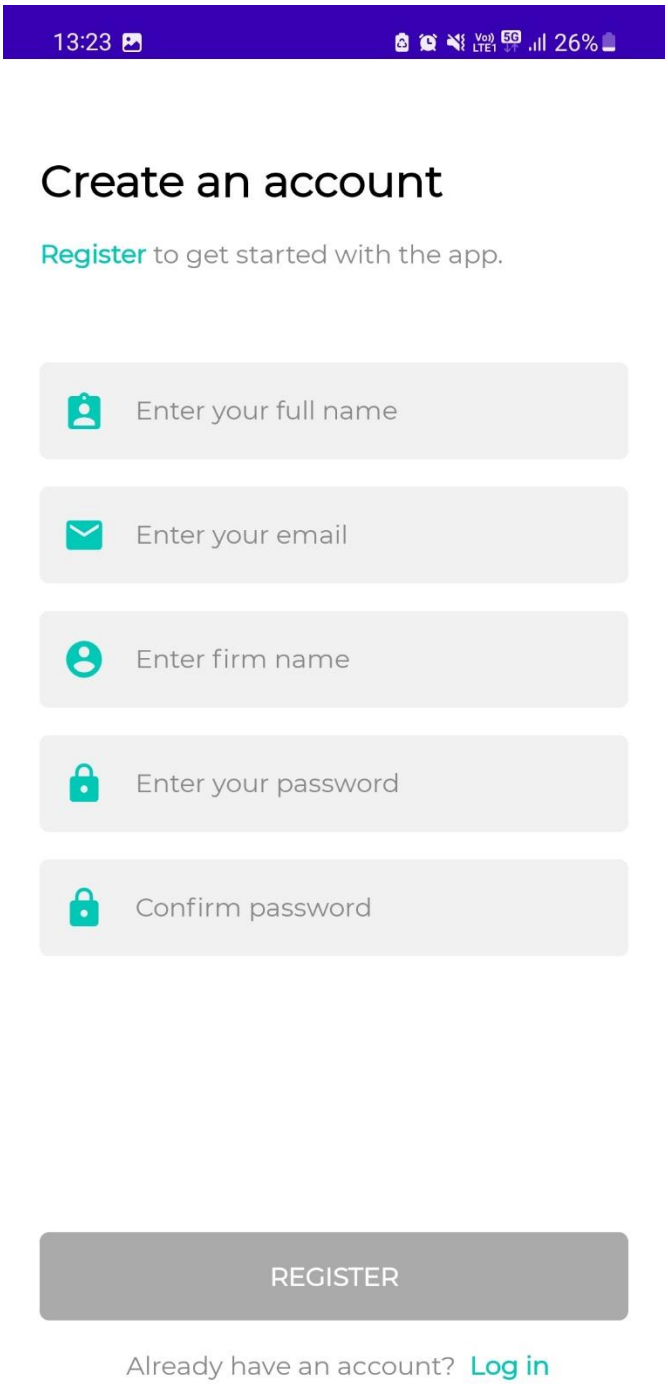
Prvi prvom otvaranju aplikacije, korisniku se prikazuje početni zaslon. Tu ima opciju prijave ako je postojeći korisnik ili registracije ako je novi korisnik. Početni zaslon prikazan je slikom 6.1.



Slika 6.1. Početni zaslon

6.2. Zaslon za registraciju korisniku

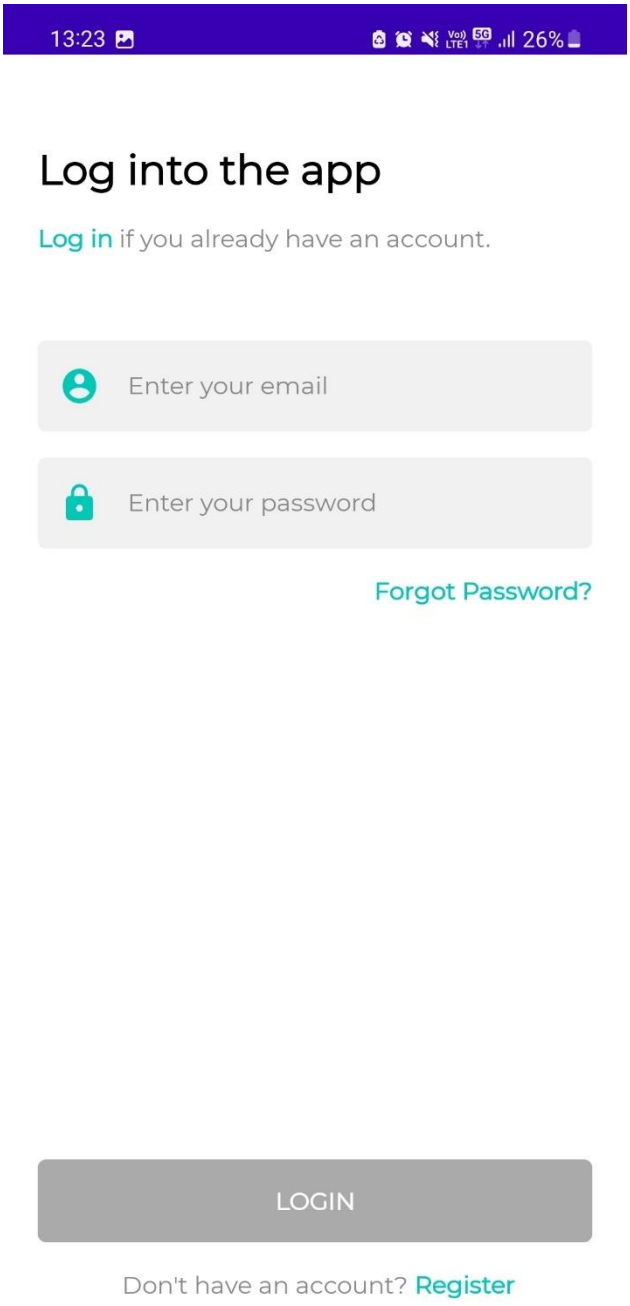
Zaslon za registraciju sastoji se od pet *TextInputLayouta*, dopstuan na [11], pet *TextInputEditTextova* te gumba za registraciju. Sadrži i mogućnost direktnog odlaska na zaslon za prijavu ako je korisnik već registriran. Zaslon za registraciju korisnika prikazan je slikom 6.2.



Slika 6.2. Zaslon za registraciju korisnika

6.3. Zaslon za prijavu

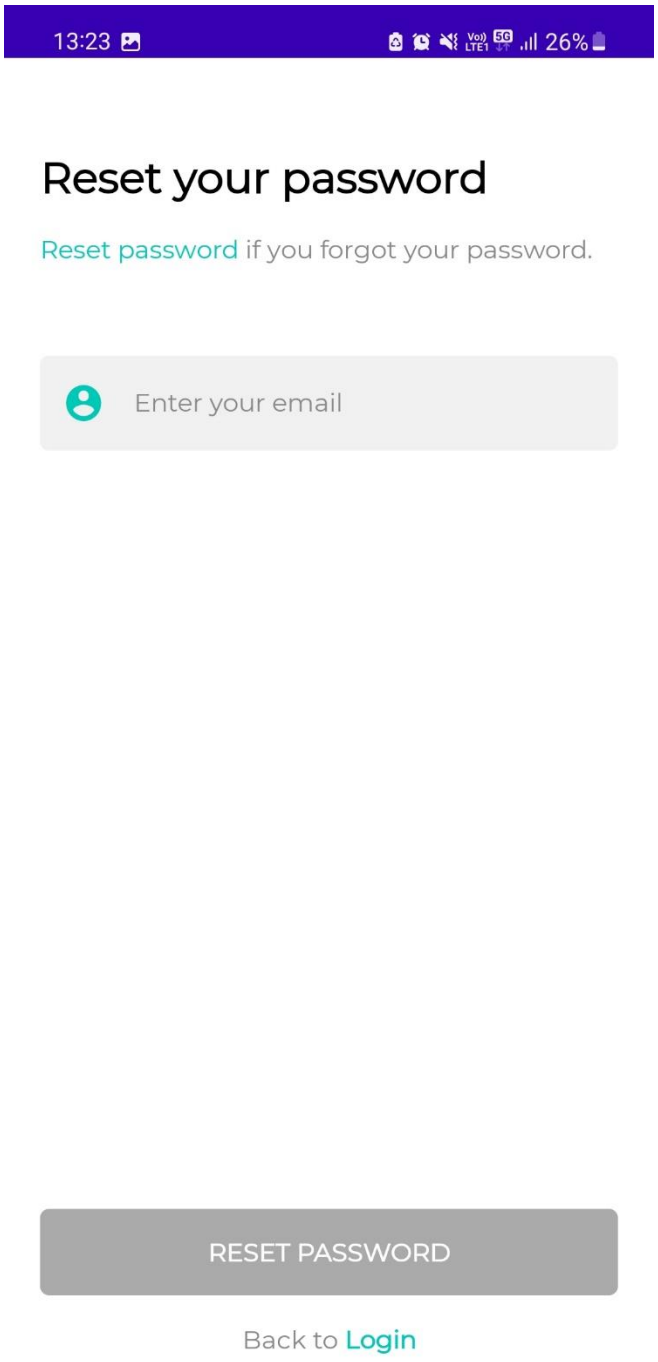
Zaslonu za prijavu se sastoji dva *TextInputLayouta*, dva *TextInputEditTextova* i gumb za prijavu. Ovaj zaslon također ima mogućnosti odlaska na zaslon za registraciju ukoliko korisnik nije prijašnje registriran ili odlaska na aktivnost za resetiranje lozinke ako je korisnik zaboravio svoju trenutnu lozinku. Zaslon za prijavu korisnika prikazan je slikom 6.3.



Slika 6.3. Zaslon za prijavu korisnika

6.4. Zaslon za resetiranje lozinke

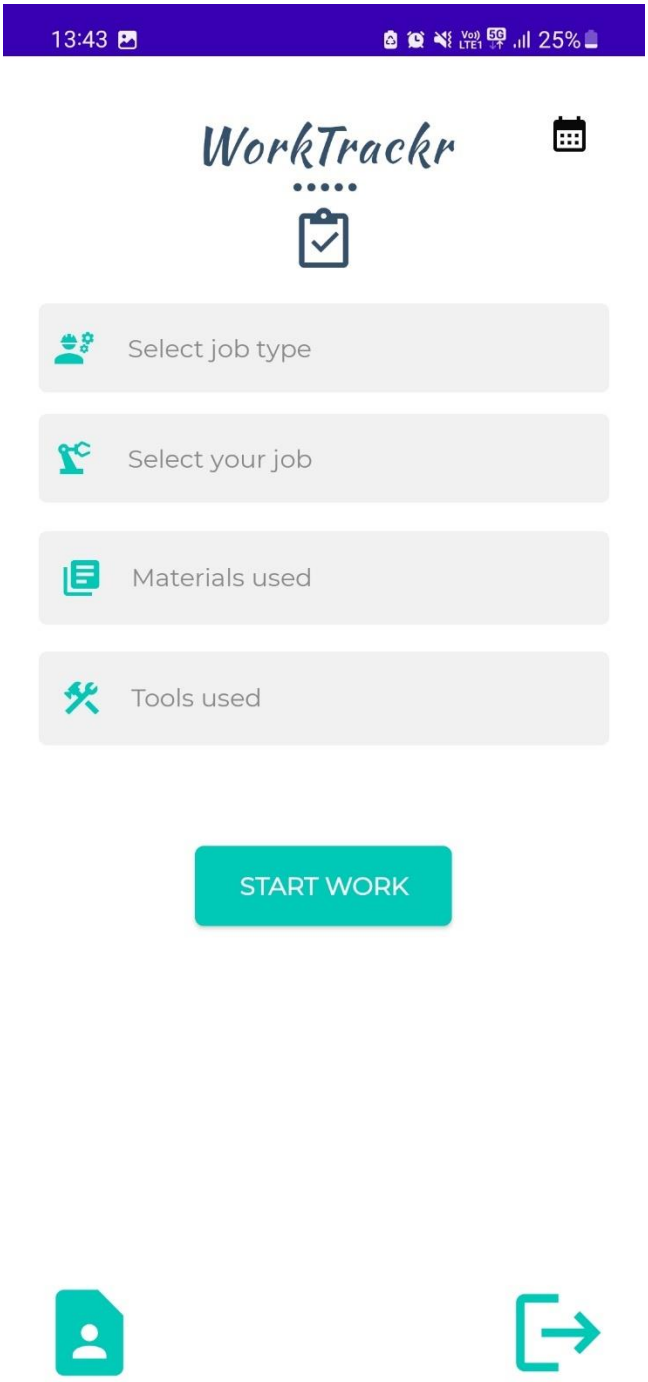
Ukoliko je korisnik zaboravio svoju lozinku, na ovome zaslonu ju može resetirati. Zaslon se sastoji od jednog *TextInputLayouta* i jednog *TextInputEditTexta* i gumba za resetiranje lozinke. Postoji mogućnost direktnog odlaska na zaslon za prijavu. Zaslon za resetiranje lozinke prikazan je slikom 6.4.



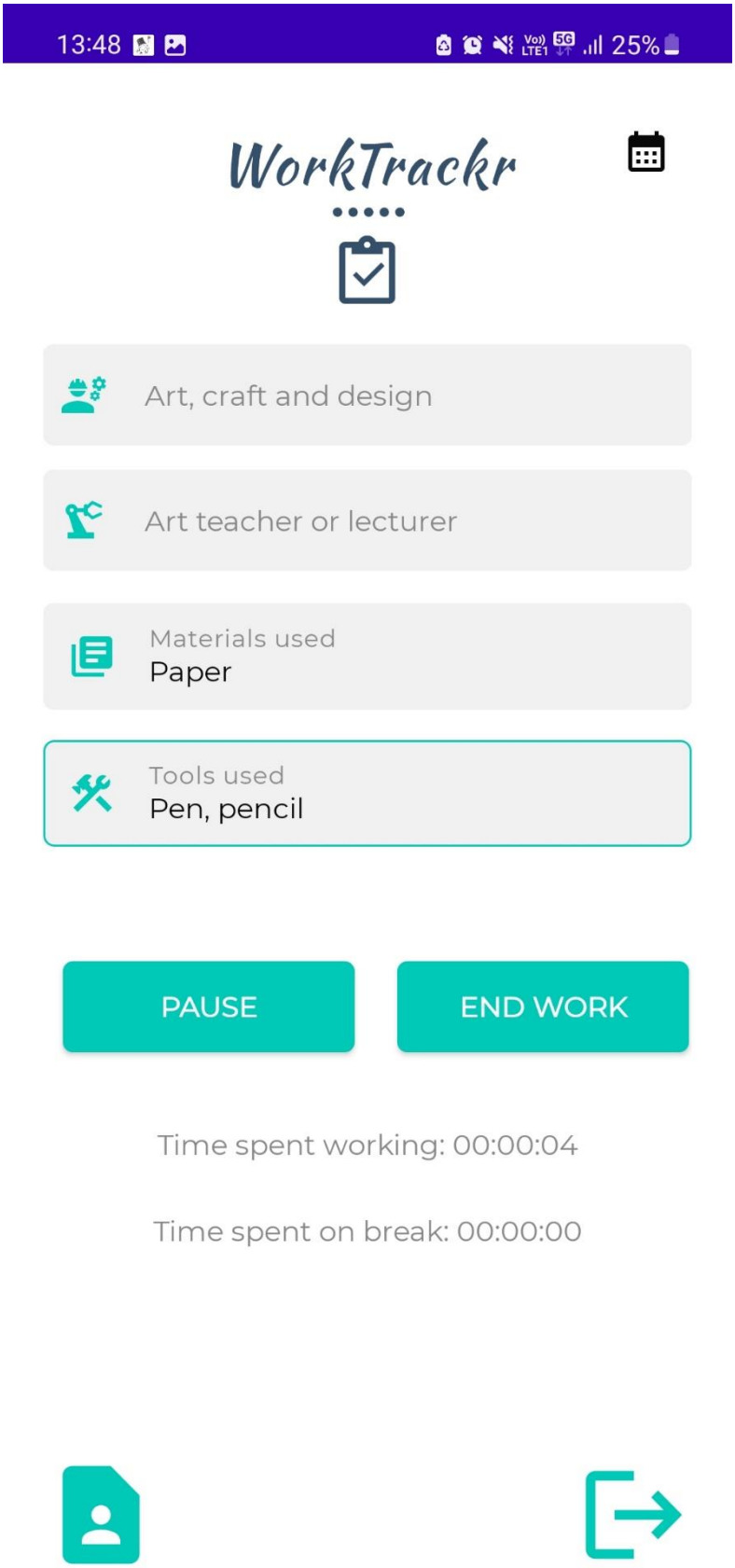
Slika 6.4. Zaslon za resetiranje lozinke

6.5. Zaslon za kreiranje radnoga naloga

Zaslon za kreiranje radnoga naloga sadrži dva klizna izbornika u kojima korisnik bira vrstu posla i pod posao, dva *TextInputLayouta* i dva *TextInputEditTextova* u koje upisuje materijale i alate te gumbove za početak rada, pauzu, kraj rada, kalendar, odlazak na zaslon kolekcije svih radnih naloga i odjavu iz aplikacije. Zaslon za kreiranje radnoga naloga prikazan je slikama 6.5. i 6.6.



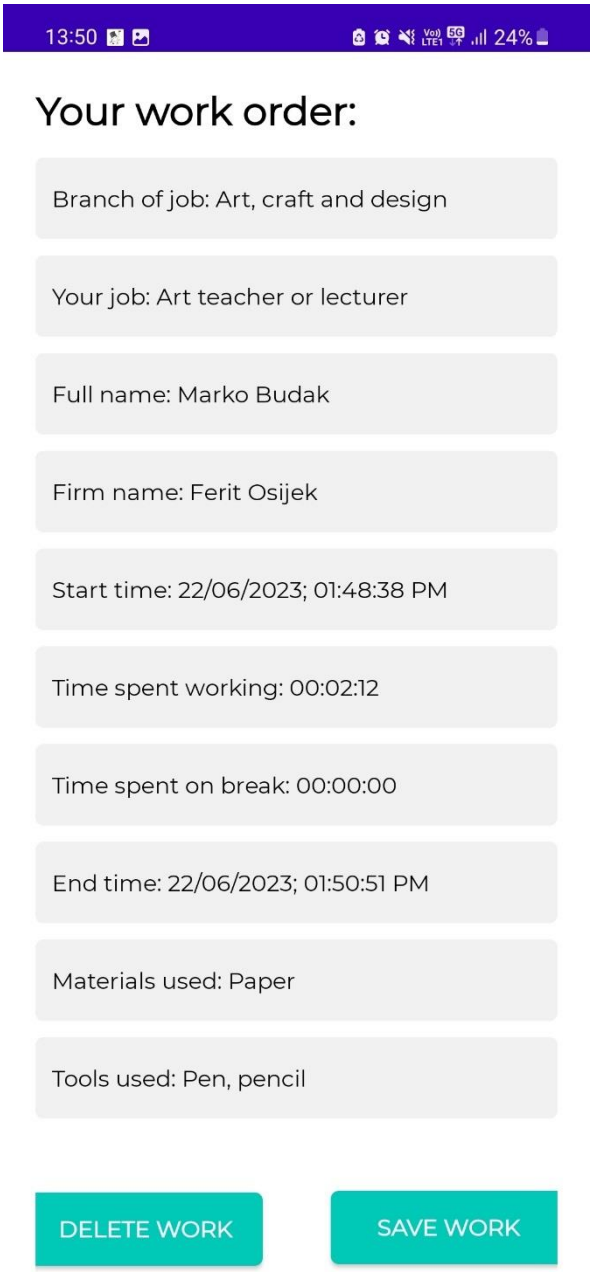
Slika 6.5. Zaslon za kreiranje radnoga naloga



Slika 6.6. Zaslon za kreiranje radnoga naloga nakon stiska na gumb "START WORK"

6.5. Zaslon za spremanje ili brisanje radnoga naloga

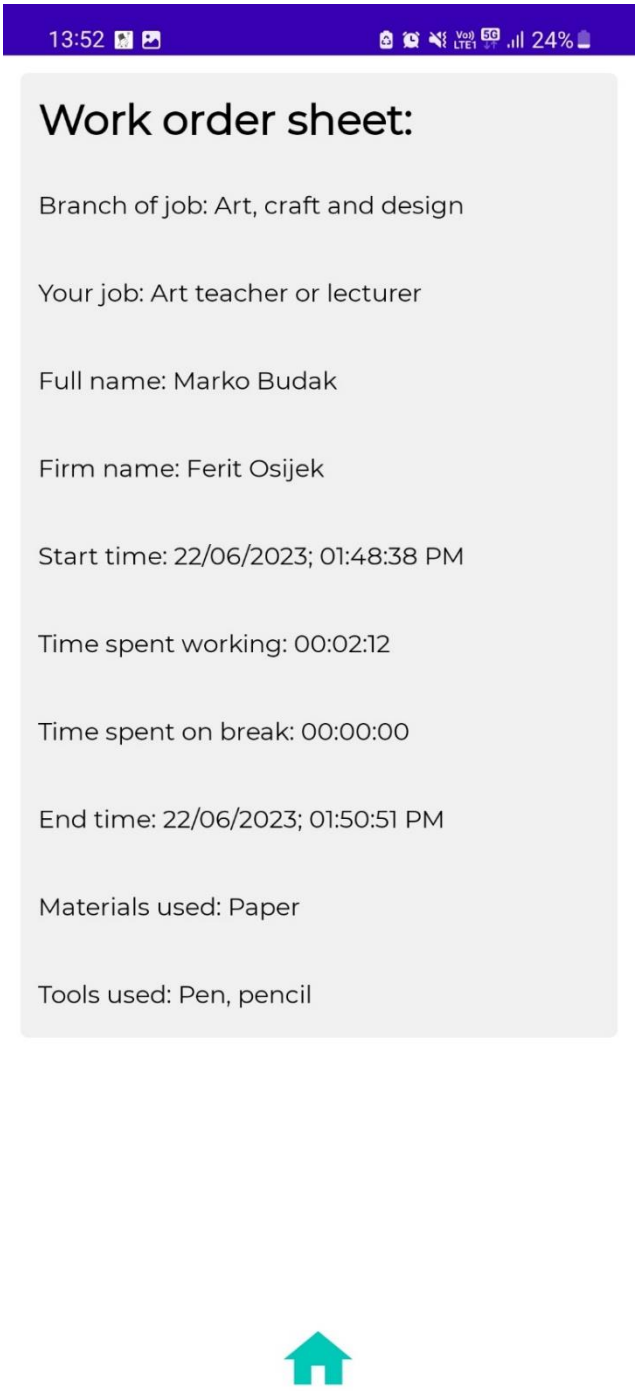
Nakon što korisnik završi s radom, dovodi ga se na zaslon za spremanje ili brisanje radnoga naloga. Ovdje korisnik ima opciju obrisati radni nalog ili spremiti radni nalog u Firestore kolekciju. Korisnik se spremanjem ili brisanjem radnoga naloga automatski vraća na zaslon za kreiranje radnoga naloga. Zaslon za spremanje ili brisanje radnoga naloga prikazana je slikom 6.7.



Slika 6.7. Zaslon za spremanje ili brisanje radnoga naloga

6.6. Zaslon kolekcije radnih naloga

Ovaj zaslon prikazuje sve radne naloge koje je korisnik napravio. Zaslon se sastoji od *RecyclerViewa* koji prikazuje njegov radni nalog i gumba za povratak na zaslon za kreiranje radnoga naloga. Zaslon kolekcije radnih naloga prikazan je slikom 6.8.



Slika 6.8. Zaslon kolekcije radnih naloga

7. ZAKLJUČAK

U ovom završnom radu istražena je i razvijena mobilna aplikacija za kreiranje radnog naloga korištenjem Android Studija i programskog jezika Kotlin. Kroz temeljito istraživanje, implementaciju i evaluaciju, uspješno smo ostvarili ciljeve završnog rada.

Korisnicima je omogućena registracija i prijava, kreiranje radnoga naloga s vrstom posla, pod posla i korištenim materijala i alatima. Izgradnja poslužiteljske aplikacije je znatno olakšana koristeći Firebase Firestore bazu podataka. Opisan je način rada prijave i registracije korisnika kroz Firebase autentifikaciju. Komunikacija i dohvaćanje podataka između klijenta i poslužitelja je opisana kroz različite aktivnosti i funkcije unutar aktivnosti. Cijela funkcionalnost i izgleda svakog zaslona je prikazana i objašnjena slikama i njenim komentarima. Aplikacija je testirana i utvrđeno je da svaki aspekt aplikacije je funkcionalan.

Ovaj završni rad pruža korisnu osnovu za daljnje poboljšanje i proširenje mobilne aplikacije za kreiranje radnog naloga. Android Studio i Kotlin su se pokazali kao moćni alati za razvoj mobilnih aplikacija, pružajući bogat skup mogućnosti i podršku za različite platforme. Kroz rad, stekli smo dublje razumijevanje procesa razvoja mobilnih aplikacija i uspješno primijenili stečeno znanje u projektu.

LITERATURA

[1] TrgovinaPlay: Timesheet - Work Hour Tracker [online], aadhk, dostpuno na: https://play.google.com/store/apps/details?id=com.aadhk.time&hl=en/ [20.6.2023.]

[2] TrgovinaPlay: Worksheet Track hours of time [online], Timechief.com, dostpuno na:https://play.google.com/store/apps/details?id=com.bakua.worktimemanager&hl=hr&gl=US/ [20.6.2023.]

[3] TrgovinaPlay: MaintainX Work Order CMMS [online], MaintainX Inc., dostpuno na:https://play.google.com/store/apps/details?id=com.commas.client&hl=en_US/ [20.6.2023.]

[4] Android Studio: Android studio [online], dopstuno na: https://developer.android.com/studio/ [20.6.2023.]

[5] Firebase [online], Google, dostupno na: https://firebase.google.com/ [21.6.2023.]

[6] Firestore [online], Google, dostupno na :https://proandroiddev.com/working-with-firestore-building-a-simple-database-model-79a5ce2692cb/ [21.6.2023.]

[7] SharedPrefrences : Save simple data with SharedPreferences [online], Developers, dostupno na: https://developer.android.com/training/data-storage/shared-preferences/ [22.6.2023.]

[8] Intent: Intent [online], Developers, dostupno na : https://developer.android.com/reference/kotlin/android/content/Intent/ [22.6.2023.]

[9] RecyclerView: Android RecyclerView in Kotlin [online], GeeksForGeeks, dostupno na: https://www.geeksforgeeks.org/android-recyclerview-in-kotlin/ [22.6.2023.]

[10] Adapter : Adapter [online] , Developers, dostupno na: https://developer.android.com/reference/kotlin/android/widget/Adapter/ [22.6.2023.]

[11] TextInputLayout : TextInputLayout [online], Developers, dostupno na: https://developer.android.com/reference/com/google/android/material/textfield/TextInputLayout/ [22.6.2023.]

SAŽETAK

Tema ovog završnog rada je mobilna aplikacija za kreiranje radnoga naloga. Cilj joj je omogućiti korisniku prijavu i registraciju, odabir vrste posla, pod posla, unos materijala i alata te mjerenje vremena provedenog na radu i vremena provedenog na pauzi. Korištena je Firebase Firestore baza podataka koji služi za spremanje svih podataka i prikazivanje radnoga naloga. Klijentski dio aplikacije odrađen je u Android Studio razvojno okruženje i programski jezik Kotlin.

Ključne riječi: Radni nalog, Firebase Firestore, Android studio, Kotlin

ABSTRACT

MOBILE APPLICATION FOR CREATING WORK ORDERS (WorkTrackr)

The topic of this final thesis is a mobile application for creating a work order. Its goal is to enable the user to log in and register, select the type of work, sub-work, enter materials and tools used, and measure time spent at work and time spent on breaks. The Firebase Firestore database was used to store all data and display the work order. The client part of the application was developed in the Android Studio development environment and the Kotlin programming language.

Keywords: Work order, Firebase Firestore, Android studio, Kotlin