SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5748

Mobilna aplikacija za prikupljanje podataka o korisničkoj aktivnosti

Mirna Baksa

Zagreb, srpanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 15. ožujka 2018.

ZAVRŠNI ZADATAK br. 5748

Pristupnik: **Mirna Baksa (0036491078)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Mobilna aplikacija za prikupljanje podataka o korisničkoj aktivnosti**

Opis zadatka:

U sklopu ovog završnog zadataka potrebno je razviti mobilnu aplikaciju za Android/iOS operacijski sustav koja će skupljati podatke o korisničkoj aktivnosti korištenja uređaja. Prije izrade aplikacije potrebno je napraviti istraživanje o mogućnostima dohvata podataka o korisniku koje pruža operacijski sustav Android/iOS. U skladu s mogućnostima potrebno je napraviti dohvat lokacije korisnika, snimanje zvuka te izradu prilagođene tipkovnice preko koje će se dohvaćati korisnikov unos. Aplikacija treba moći prikupljati podatke u pozadinskom načinu rada. Također je potrebno razraditi model podataka i način na koji će se spremati dohvaćeni podaci kako bi bili pogodni za dalju analizu.
Programski kod je potrebno komentirati i pri pisanju pratiti neki od standardnih stilova. Kompletnu aplikaciju postaviti na repozitorij Github.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:

Izv. prof. dr. sc. Mile Šikić

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

S razvojem tehnologije, prikupljanje korisničkih podataka u raznim mobilnim i web aplikacijama postalo je uobičajeno i nužno te gotovo svaka aplikacija na današnjem tržištu skuplja od korisnika neku vrstu podataka.

Podaci koji se skupljaju mogu se grubo podijeliti u dvije kategorije:

1. eksplicitni,
2. implicitni.

Eksplicitnim podacima podrazumijevamo podatke dobivene izravno od korisnika. To mogu biti osobni podaci poput imena, adrese elektroničke pošte, spola, dobi i slično, dobivenih ispunjavanjem raznih obrazaca u aplikaciji (npr. obrasca za registraciju ili anketa). Ovakva vrsta podataka smatra se valjanom i pouzdanom, s obzirom da je dobivena izravno od korisnika, pa ne zahtjeva dodatne analize i interpretacije.

S druge strane, *implicitni* podaci se skupljaju indirektno kroz analizu korisnikove interakcije s aplikacijom. Ovi podaci mogu uključivati podatke o uređaju, načinu i duljini korištenja aplikacije i slično. Za razliku od eksplicitnih podataka, implicitni podaci zahtjevaju daljnje analize kako bi nam njihova interpretacija donijela željene informacije.

U sklopu ovog završnog rada razvijena je aplikacija koja implicitno prikuplja podatke s mobilnog uređaja, nemaštovito nazvana *Collector* (*Prikupljač*). Aplikacija je napravljena generički i moguće ju je koristiti u različite svrhe, no glavna motivacija je bila izrada aplikacije koja bi pratila aktivnost pacijenata. Razvoj ovakvih aplikacija je u usponu, pogotovo s razvojem tzv. *wearable* uređaja, tj. uređaja koji se mogu nositi (pametni satovi, narukvice itd.). Analizom informacija sakupljenih s uređaja moglo bi se pratiti stanje pacijenta na dnevnoj bazi i lakše donositi personalizirane dijagnoze. Naravno, ovakva tehnologija je preširoka za okvir završnog rada, pa ovaj rad samo prikazuje implementaciju sustava koji prikuplja, sprema i prikazuje podatke. Daljnja analiza tih podataka i izvlačenje praktičnog znanja ostavljena je za neki veći rad. Uz

mobilnu, razvijena je i jednostavna web aplikacija za prikaz podataka prikupljenih s uređaja.

Ono što je bitno spomenuti kod implicitnog prikupljanja podataka je privatnost i anonimnost korisnika, što je sve aktualnija tema današnjice. Većina korisnika danas nije ni svjesna o količini informacija koje se prikupljaju o njima i njihovoj aktivnosti, iako se kroz pritiske medija i društvenih mreža i to počelo mijenjati, te su aplikacije primorane tražiti dozvole za prikupljanje podataka od korisnika - jednostavan i praktičan primjer je traženje dozvole za omogućivanje prikupljanja kolačića (engl. *cookies*). U ovom pogledu razvijena aplikacija pretpostavlja da korisnik želi da se o njemu skupljaju podaci, s obzirom da je aplikacija rađena isključivo i eksplicitno u tu svrhu.

Rad je podijeljen u tri dijela. U prvom dijelu razrađeno je oblikovanje sustava - konkretan opis zadatka, odabir operacijskog sustava za mobilnu aplikaciju te razvojne okoline za web aplikaciju. Dan je model podataka. U drugom dijelu detaljno je objašnjena implementacija i mobilne i web aplikacije. Treći dio daje rezultate testiranja potrošnje baterije mobilne aplikacije.

2. Oblikovanje sustava

2.1. Opis zadatka

U sklopu ovog zadatka bilo je potrebno razviti mobilnu aplikaciju za Android ili iOS operacijski sustav koja sakuplja podatke o korisničkoj aktivnosti korištenja uređaja. Prije izrade aplikacije bilo je nužno istražiti mogućnosti dohvata podataka o korisniku koje pruža određeni operacijski sustav, te u skladu s time odabrati operacijski sustav za implementaciju.

U skladu s mogućnostima trebalo je napraviti dohvat lokacije korisnika, snimanje zvuka, te izraditi prilagođenu tipkovnicu preko koje će se dohvaćati korisnikov unos. Glavni zahtjev je da je sve podatke treba prikupljati u pozadinskom načinu rada.

Za sve prikupljene podatke bilo je potrebno razraditi model podataka te način na koji će se oni spremati kako bi bili pogodni za daljnju analizu.

2.2. Korištene tehnologije i alati

2.2.1. Mobilna aplikacija

Odabir mobilnog operacijskog sustava - Android

Kao operacijski sustav za razvoj mobilne aplikacije odabran je Android OS umjesto iOS-a.

Android je operacijski sustav otvorenog koda (engl. *open-source*) razvijen od strane Google-a baziran na Linux jezgri. Upravo mu činjenica da je otvorenog koda daje fleksibilnost, i za korisnika i za developera. Aplikacije za Android su pisane pomoću Android Software Development kit-a (SDK), a službeni programski jezik od svibnja 2017. godine je Kotlin, iako se još uvijek velikim dijelom koristi programski jezik Java. Java može biti kombinirana sa C/C++-om čime se dobiva na efikasnosti i brzini aplikacije.

Razlozi odabira Android OS-a su djelomično čisto praktični - razvoj Android aplikacije moguć je na bilo kojem operacijskom sustavu (Windows, Linux ili Mac), dok je za iOS aplikacije potreban Mac OS. Također, za skupljanje osjetljivih podataka od uređaja (što je i cilj ove aplikacije) Android daje veću fleksibilnost. Kao programski jezik korištena je Java.

Kao razvojno okruženje korišten je Android Studio.

2.2.2. Web aplikacija

Za izradu web aplikacije korišten je programski jezik PHP i radno okruženje (engl. *framework*) Laravel u kombinaciji s Composer-om. Composer je alat za upravljanje paketima (engl. *dependency manager*) u aplikacijama napisanim u PHP programskom jeziku. Pomoću alata Composer moguće je instalirati i ažurirati sve pakete koji su potrebni za aplikaciju koja se razvija.

Za razvoj je odabran Laravel zbog svoje ekspresivnosti i jednostavnosti. On kao radno okruženje kombinira najbolje funkcionalnosti drugih web radnih okruženja kao što su Ruby on Rails, ASP.NET MVC i Sinatra.

Kao razvojno okruženje korišten je Visual Studio Code.

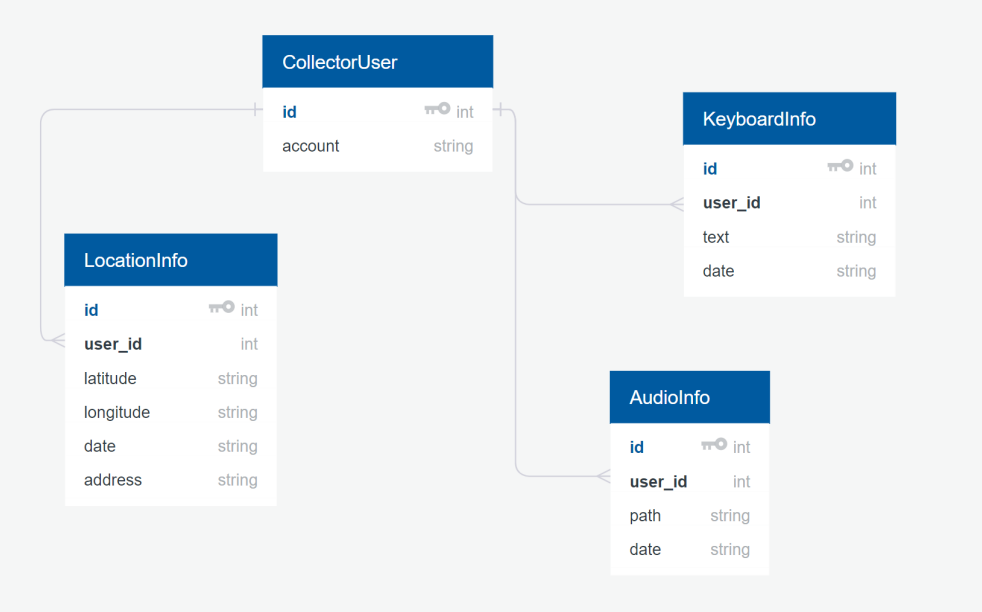
2.3. Model podataka

Razrada efikasnog modela podataka ponekad je izazov. Međutim, s obzirom da u ovoj aplikaciji nema mnogo međuovisnosti između podataka (jedan zapis o lokaciji je nezavisan o jednom zapisu zvuka), to se nije predstavilo kao veći problem. Podaci su zavisni jedino u pogledu korisnika od kojeg je podatak sakupljen.

ER dijagram baze podataka aplikacije prikazan je na slici 2.1.

Svi korisnici nalaze se u tablici *CollectorUser* i jedinstveno su identificirani *id*-jem. Polje *account* predstavlja ime računa s kojim se korisnik prijavljuje na početku korištenja aplikacije.

Zapisi o lokaciji, unosu teksta i zvuku također su jedinstveno identificirani *id*-jem, a kao strani ključ imaju *user_id* koji referencira tablicu *CollectorUser*. Uz to, zapisi o lokaciji (*LocationInfo*) sadrže geografsku širinu i visinu, datum zapisa te adresu dobivenu iz zabilježene geografske lokacije. Uneseni tekst bilježi se u polju *text* tablice *KeyboardInfo*, a bilježi se i datum zapisa. Zapis o snimljenom zvuku (tablica *AudioInfo*) sadrži putanju do audio datoteke na disku te datum zapisa.



Slika 2.1: ER model baze

Ovakav model podataka omogućava povezivanje zapisa s korisnikom, ali opet ne sadrži toliko međuovisnosti između različitih zapisa da bi njihov dohvat i obrada bili kompliciranji.

3. Mobilna aplikacija

3.1. Općenito

S obzirom da tema ovog rada nije općeniti opis razvoja mobilne aplikacije, sav će tekst biti fokusiran na praktičnu izvedbu konkretne aplikacije. Za više informacija o strukturi Android aplikacije i razvoju iste, čitatelja se upućuje na službene stranice za Android developere [1].

Implementacija aplikacije dostupna je na GitHub-u [2].

Struktura aplikacije

Aplikacija se sastoji od jednog *Activity-a* i tri *Service-a*.

Activity u Androidu predstavlja jedan ekran s korisničkim sučeljem. Razred *Activity* je ključna komponenta Android aplikacija, a način na koji su aktivnosti međusobno povezane čini osnovni dio strukture aplikacije.

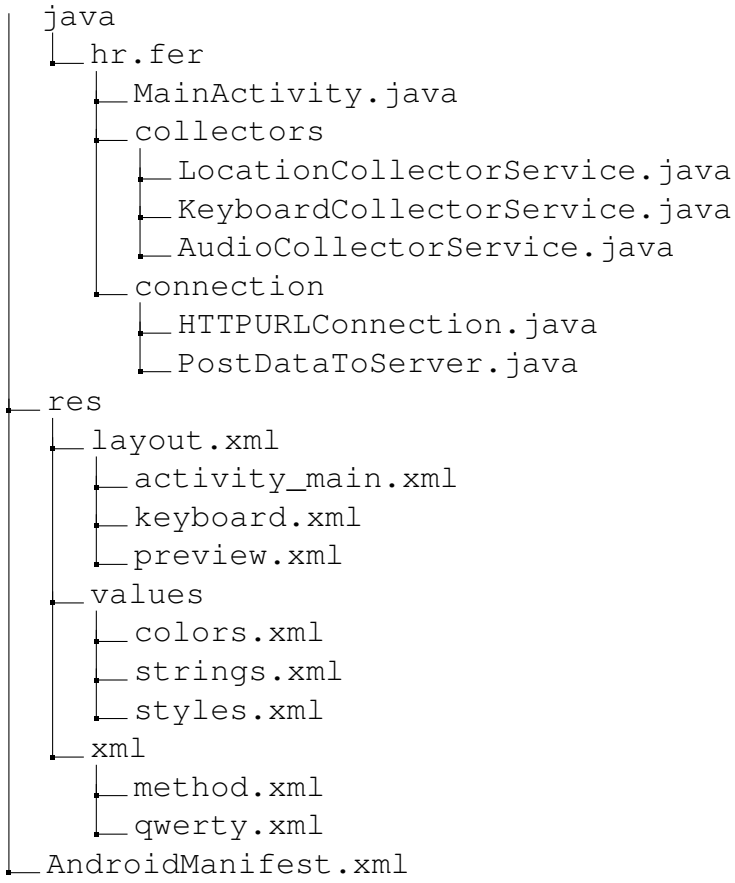
S druge strane, *Service* je komponenta aplikacije koja izvodi operacije u pozadini i ne pruža korisničko sučelje.

S obzirom da je glavni zahtjev razvijane aplikacije da se podaci skupljaju u pozadinskom načinu rada, to će biti ostvareno preko *Service-a*. Za svaki od podataka koje trebamo sakupljati (lokacija, tekst, zvuk) implementiran je jedan *Service* koji prikuplja podatke i šalje ih na vanjski server. Jedan jedini *Activity* koji je implementiran u aplikaciji služi za dobivanje dozvola, odabir korisničkog računa te uključivanje/isključivanje pojedinog prikupljača podataka.

Struktura direktorija izvornog koda aplikacije dana je ispod. Objašnjenje uloge razreda *MainActivity.java* dano je u odjeljku 3.4. Objašnjenja uloga razreda paketa *collectors* dana su respektivno u odlomcima 3.3, 3.4, 3.5, a paketa *connection* u odlomku 3.6.

[1] `https://developer.android.com/`

[2] `https://github.com/mirnabaksa/CollectorAndroid`



Slika 3.1: Struktura aplikacije

Uz izvorne kodove, u direktoriju *res* nalaze se *.xml* datoteke koje definiraju resurse aplikacije te datoteka *AndroidManifest.xml*.

Android manifest

Svaka Android aplikacija obavezno sadrži *AndroidManifest.xml* datoteku koja navodi ključne informacije o aplikaciji koje potom koriste Android alati, Android operacijski sustav te Google Play.

Manifest između ostalog opisuje: ime paketa aplikacije, popis komponenata aplikacije, popis dozvola koje aplikacija koristi, popis zahtjeva na hardware i software uređaja itd.

Manifest razvijene aplikacije sadrži:

1. **ime paketa aplikacije**

```
<manifest xmlns:android=
   "http://schemas.android.com/apk/res/android"
   package="hr.fer">
```

Implementacija je smještena u *hr.fer* paket.

2. **popis komponenata aplikacije**

```
<activity android:name="hr.fer.MainActivity">
<service
   android:name="hr.fer.collectors.LocationCollectorService">
<service
   android:name="hr.fer.collectors.KeyboardCollectorService">
<service
   android:name="hr.fer.collectors.AudioCollectorService">
```

Upravo ovdje definiramo jedan activity i tri service-a.

3. **popis dozvola koje aplikacija koristi**

Tablica 3.1: Korištene dozvole

Dozvola	Opis
ACCESS_FINE_LOCATION	Dohvat precizne lokacije
ACCESS_COURSE_LOCATION	Dohvat približne lokacije
ACCESS_NETWORK_STATE	Dohvat informacija o stanju mreže
INTERNET	Spajanje na Internet
RECORD_AUDIO	Snimanje zvuka
GET_ACCOUNTS	Dohvat korisnikovih računa
WRITE_EXTERNAL_STORAGE	Pisanje po disku uređaja

Primjer retka za traženje dozvole u manifestu:

```
<uses-permission
   android:name="android.permission.INTERNET" />
```

3.2. Početni ekran

Početni ekran implementiran je u razredu *MainActivity.java*. Iako grafičko sučelje nije bilo jedno od potrebnih funkcionalnosti aplikacije, ovakvim je pristupom dana veća fleksibilnost u korištenju.

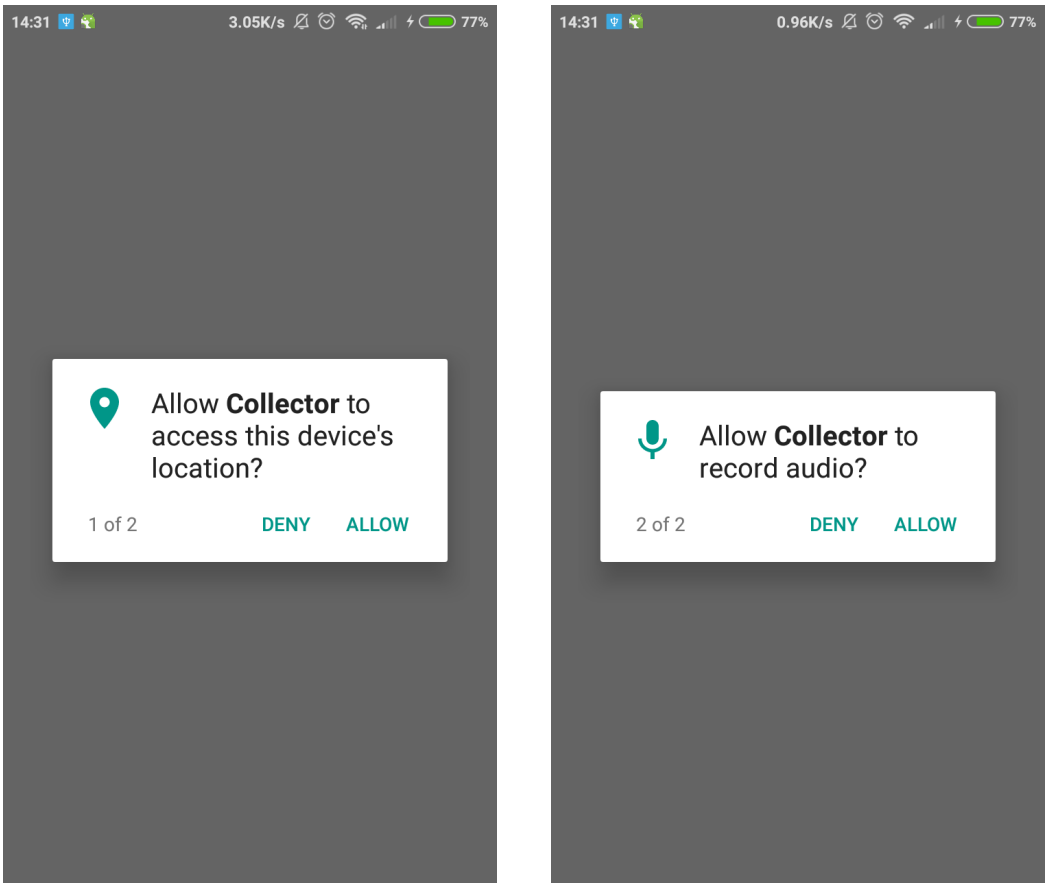
```
public class MainActivity extends AppCompatActivity{ ... }
```

MainActivity ima tri odgovornosti: dobivanje potrebnih dozvola od korisnika, odabir korisničkog računa za korištenje u aplikaciji te pružanje odabira o uključivanju/isključivanju pojedinog prikupljača podataka (odnosno *Service-a* koji prikuplja određeni podatak).

Dobivanje dozvola

Iako je već u manifestu bilo navedeno koje dozvole koristi aplikacija, neke od tih dozvola su tzv. opasne dozvole (engl. *dangerous permission*). Od verzije Androida 6.0 ili više (API level 23 ili više), korisnici kod instalacije aplikacija više nisu obaviješteni o dozvolama koje traži aplikacija. Aplikacije za takve dozvole moraju tražiti dopuštenje od korisnika prilikom prvog pokretanja aplikacije.

Opasne dozvole u ovoj aplikaciji su dozvole za lokaciju i snimanje zvuka.



(a) Lokacija **(b)** Snimanje zvuka

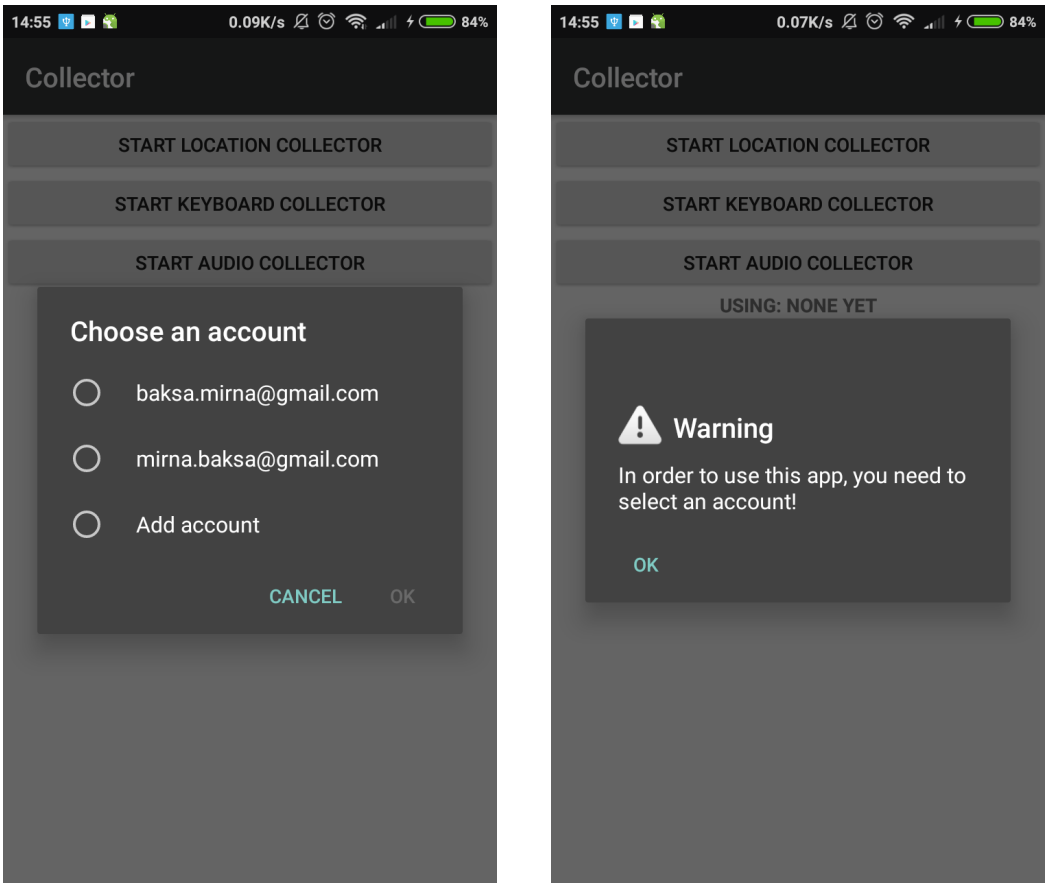
Slika 3.2: Opasne dozvole

Odabir korisničkog računa

Za identificiranje korisnika odabran je najjednostavniji mogući pristup. S obzirom da zahtjev aplikacije nije da korisnik vidi svoje skupljene podatke (iako bi to proširenje aplikacije bilo dobrodošlo u budućnosti), zasad nije bilo potrebno implementirati funkcionalnost registracije/prijave u aplikaciju.

Međutim, idenfikacija korisnika je još uvijek potrebna kako bi znali odrediti koji zapis pripada kojem korisniku aplikacije. Za to su iskorišteni Google računi - sigurno je da je Android uređaj povezan barem s jednim, ako ne i više računa.

Nakon što korisnik odobri potrebne dozvole, otvara se početni ekran aplikacije te se od korisnika traži da odabere račun kojim će ga aplikacija dalje identificirati. Odabir računa je obavezan, pa ako korisnik pritisne gumb "Cancel", pojavljuje se prikladno upozorenje i opet se pojavljuje dijalog za odabir računa.



(a) Dijalog za odabir **(b)** Upozorenje

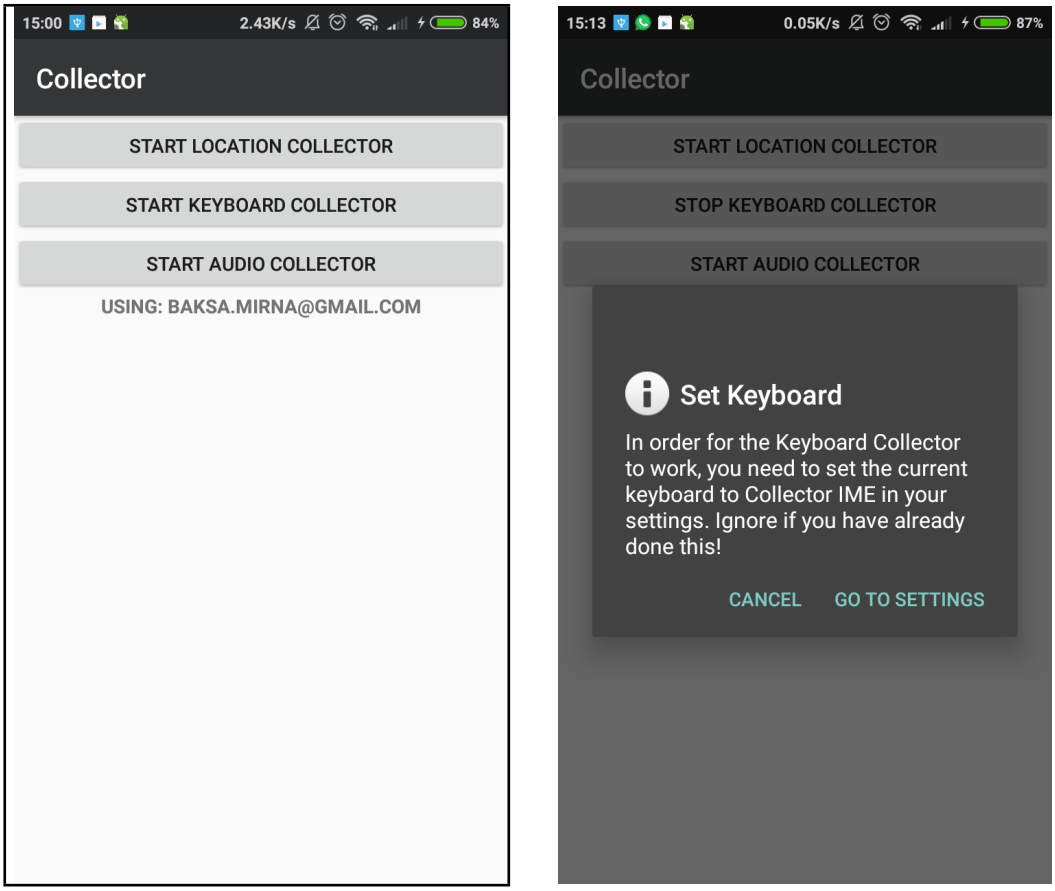
Slika 3.3: Odabir računa

Uključivanje i isključivanje prikupljača

Nakon dobivanja potrebnih dozvola i odabira računa, konačno se otvara početni ekran aplikacije.

Aplikacija nudi odabir koji će se podaci sakupljati u kojem trenutku, a tu funkcionalnost ostvaruje s tri gumba pritiskom na koje se pokreće ili gasi pojedini *Service* koji prikuplja podatke. Uz to, naveden je i račun kojim se identificira trenutni korisnik.

S obzirom da je za prikupljanje utipkanog teksta razvijena zasebna tipkovnica, da bi prikupljač teksta radio potrebno je u postavkama mobilnog uređaja odabrati tu tipkovnicu kao trenutnu. Pritiskom na gumb *"Start keyboard collector"* otvara se dijalog koji korisniku nudi prečac (engl. *shortcut*) do postavki uređaja.



(a) Početni ekran **(b)** Promjena tipkovnice

Slika 3.4: Početni ekran i odabir tipkovnice

Implementacija

U ovom je odjeljku dan kratak pregled metoda koje sadrži *MainActivity.java* te njihova funkcionalnost.

Deklaracija metode	Opis funkcionalnosti
`@Override` `protected void onCreate(Bundle savedInstanceState);`	Nadjačana metoda osnovnog razreda (AppCompatActivity). Pokreće traženje dozvola ako one već nisu omogućene. Pokreće dijalog za odabir računa ukoliko to već nije napravljeno.
`@Override` `protected void onSaveInstanceState(Bundle savedInstanceState);`	Nadjačana metoda osnovnog razreda. Osigurava da se kod promjene orijentacije ekrana ponovno ne pojavljuje dijalog za odabir računa.
`@Override` `protected void onActivityResult(int requestCode, int resultCode, Intent data);`	Procesuira rezultat Intenta koji dohvaća račun od korisnika. Ukoliko je dijalog bio otkazan, prikazuje dijalog s upozorenjem (Slika 3.3b).
`private void configureIntents();`	Stvara i inicijalizira Intent-e za pokretanje Service-a.
`private void configureButtons();`	Konfigurira gumbe za pokretanje i gašenje prikupljača.
`public void checkPermission();`	Traži opasne dozvole od korisnika.

`private void manageService(Intent intent, boolean start);`	Pokreće ili zaustavlja (ovisno o zastavici start) service pokrenut datim Intent-om.

Tablica 3.2: Metode u MainActivity.java

3.3. Lokacija

Zapisi o lokaciji prikupljaju se u razredu *LocationCollectorService*.

```
public class LocationCollectorService extends Service { ... }
```

Lokacija je jedan od najčešćih podataka koji se prikupljaju s mobilnih uređaja, pa shodno tome Android nudi lepezu mogućnosti - od biranja frekvencije osvježavanja lokacije, mogućnosti čuvanja potrošnje baterije, dobivanja preciznih adresa i slično. Ove su značajke definirane primjerkom *LocationRequest* razreda. U aplikaciji je definirano sljedeće:

```
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(10 * 60 * 1000);
mLocationRequest.setFastestInterval(50 * 60 * 1000);
mLocationRequest.setSmallestDisplacement(10.0f);
mLocationRequest.setPriority(
  LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
```

Metode *setInterval()* i *setFastestInterval()* definiraju frekvenciju osvježavanja lokacije. Prva metoda definira preferirani interval, a druga najbržu frekvenciju koju aplikacija može/želi podnijeti. U ovom slučaju to su intervali od 10 odnosno 5 minuta (ovi intervali su proizvoljni i mogu se prilagoditi potrebama aplikacije).

Metoda *setSmallestDisplacement()* osigurava da se informacije o lokaciji dobivaju samo ako je uređaj prošao udaljenost od 10 metara (proizvoljno). Ovim se osigurava da se sustav ne pretrpava informacijama - npr. ukoliko je aplikacija upaljena kad korisnik spava i uređaj se ne miče, nisu potrebni zapisi svakih 10 minuta.

Metodom *setPriority()* biramo između opcija:

Tablica 3.3: Opcije dohvata lokacije

Opcija	Razina preciznosti
PRIORITY_HIGH_ACCURACY	Najpreciznije trenutno dostupne informacije o lokaciji
PRIORITY_BALANCED_POWER_ACCURACY	Razina kvarta(engl. *block*)
PRIORITY_LOW_POWER	Razina grada
PRIORITY_NO_POWER	Najbolja moguća preciznost bez dodatne potrošnje energije

U aplikaciji koristimo opciju *PRIORITY_BALANCED_POWER_ACCURACY* kao kompromis između točnosti i uštede baterije.

Nakon definiranja svih potrebnih specifikacija *LocationRequest*-a, redovna ažuriranja lokacije dobivamo pozivom *requestLocationUpdates()* metode razreda *FusedLocationProviderClient*:

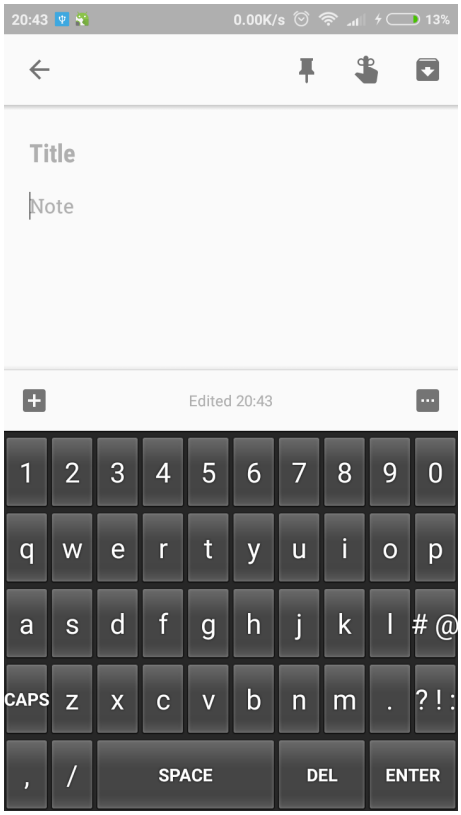
```
FusedLocationProviderClient mFusedLocationClient =
    LocationServices.getFusedLocationProviderClient(this);
mFusedLocationClient.requestLocationUpdates(mLocationRequest,
  mLocationCallback,null);
```

FusedLocationProviderClient kod dojave promjene lokacije zove metodu *LocationCallback.onLocationChanged()* objekta predanog kod poziva metode *requestLocationUpdates()* .

LocationCallback je sučelje čijim implementiranjem metode *onLocationChanged()* možemo dohvatiti i obraditi informaciju o prikupljenoj lokaciji. Upravo se u implementaciji te metode dohvaća lokacija, pripremaju parametri za slanje na server te naposljetku inicira slanje na server.

3.4. Tipkovnica

U sklopu ovog dijela zadatka razvijena je prilagođena tipkovnica čija konfiguracija je dana u datotekama *method.xml* i *keyboard.xml*. Izgled tipaka je definiran u datoteci *qwerty.xml*. Sve navedene datoteke nalaze se u direktoriju *res* (vidi Sliku 3.1).



Slika 3.5: Izgled tipkovnice

Prikupljanje podataka o unesenom tekstu vrši se u razredu *KeyboardCollectorService*.

```
public class KeyboardCollectorService extends
    InputMethodService implements
    KeyboardView.OnKeyboardActionListener { ... }
```

Razred nasljeđuje *InputMethodService* što omogućuje izradu prilagođenog IME-a (*input method*) te implementira sučelje *OnKeyboardActionListener* preko kojeg se dobivaju informacije o unesenom tekstu. Ovdje je razred *KeyboardCollectorService* sudionik (konkretni promatrač) oblikovnog obrasca Promatrač[3], u kojem je subjekt upravo tipkovnica.

[3] https://en.wikipedia.org/wiki/Observer_pattern

Samo prikupljanje podataka inspirirano je principom priručne (engl. *cache*) memorije. Razred *KeyboardCollectorService* obaviješten je o svakom pritisku tipke na tipkovnici pozivom metode *onKey()*.

```
public void onKey(int primaryCode, int[] keyCodes);
```

Ukoliko je pritisnuta alfanumerička tipka (A-Z, 0-9) ili neki od interpunkcijskih znakova, taj se zapis bilježi pomoću primjerka razreda *StringBuilder* koji služi kao svojevrsni privremeni spremnik (engl. *buffer*). *StringBuilder* omogućuje korištenje Stringova kao promjenjivih (engl. *mutable*) objekata, što nije moguće kod korištenja osnovne *String* klase. Ovako se izbjegava stvaranje nepotrebnih objekata i samim time dobiva na efikasnosti.

Nakon što duljina zabilježenog zapisa bude veća od 100, taj se zapis upisuje u pomoćnu *cache* datoteku. Duljina od 100 znakova odabrana je proizvoljno i može se prilagoditi zahtjevima aplikacije.

Utipkani podaci se čitaju iz datoteke svaku minutu te šalju na server. Interval je opet proizvoljan. Za periodično slanje podataka iskorišten je razred *TimerTask* u kombinaciji s razredom *Handler* koji služi za izvođenje operacija na zasebnoj dretvi.

```
private Handler handler = new Handler();

private class CacheTimer extends TimerTask{

      @Override
      public void run() {
         handler.post(new Runnable() {
            @Override
            public void run() {
               ...

         });
      }
   }
```

Korištenjem priručne memorije u kombinaciji s razredom *StringBuilder* omogućeno je efikasnije prikupljanje podataka. S obzirom da su pisanje u datoteku i slanje na server dvije potencijalno skupe operacije, ovako se postiže kompromis.

3.5. Zvuk

Prikupljanje zvuka vrši se u razredu AudioCollectorService.

```
public class AudioCollectorService extends Service { ... }
```

Za snimanje zvuka koristimo razred *MediaRecorder*. Kao i kod dohvata lokacije, snimanje zvuka je česta potreba pa je to u Androidu svedeno na nekoliko linija koda. Prije početka snimanja definiramo sljedeće parametre audio zapisa:

- **izvor zvuka** - mikrofon mobilnog uređaja
- **format audio zapisa** - .3gp. Odabran je ovaj format jer je generalno manje veličine od ostalih formata (npr .mp4 formata).
- **ime zapisa** - u formatu "audio-trenutno_vrijeme_u_ms.3gp". Ovakav zapis osigurava jedinstvenost imena zapisa.

```
MediaRecorder mRecorder = new MediaRecorder();
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mRecorder.setOutputFile(mFileName);
```

Za početak snimanja potreban je poziv metoda *prepare()* i *start()* primjerka razreda *MediaRecorder*, a za kraj snimanja metode *stop()*.

Snimljena datoteka nalazi se u internoj memoriji uređaja i spremna je za prijenos na server.

3.6. Slanje podataka na server

Za slanje podataka na server iskorištena je funkcionalnost HTTP zahtjeva.

Hypertext Transfer Protokol (HTTP) je aplikacijski protokol koji čini temelj za osnovnu komunikaciju na Internetu. HTTP funkcionira kao zahtjev - odgovor (engl. *request - response*) protokol između klijenta i servera.

Klijent je u ovom slučaju mobilna aplikacija, dok je server web aplikacija također razvijena u okviru ovog završnog rada. Sva funkcionalnost obrade podataka i njihovog spremanja u bazu podataka implementirana je u web aplikaciji.

Pozadinsko slanje podataka (odnosno zahtjeva) na server izvodi razred *PostDataToServer*.

```
class PostDataToServer extends AsyncTask<Void, Void, Void>
```

Razred nasljeđuje *AsyncTask* koji omogućuje izvođenje operacija u pozadini bez potrebe za ručnim korištenjem dretvi. Ovaj je razred idealan za izvođenje kratkih operacija, a slanje HTTP zahtjeva je upravo to.

Razred *PostDataToServer* u konstruktoru prima putanju do servera i mapu s parametrima HTTP zahtjeva. U konstruktoru se stvara novi primjerak razreda *HTTPURLConnection* koji je zadužen za proces stvaranja HTTP zahtjeva i njegovog slanja na server. Operacija koja se treba izvoditi navodi se u metodi *doInBackground()*.

```
public PostDataToServer(String serverPath,
   HashMap<String,String> postDataParams);

@Override
 protected Void doInBackground(Void... params);
```

Razred *HTTPURLConnection* u metodi *send()* otvara vezu prema serveru, konfigurira zaglavlje i tijelo zahtjeva te naposljetku šalje HTTP POST zahtjev na server.

```
public String send(String path, HashMap<String, String>
   params);
```

Pokažimo primjer slanja podataka na server. Pretpostavimo da šaljemo tekstualni zapis.

```
//priprema parametara
HashMap<String, String> postDataParams = new HashMap<>();
postDataParams.put("account", "baksa.mirna@gmail.com");
postDataParams.put("text", "Mobilna aplikacija za
   prikupljanje podataka ");
postDataParams.put("date", "Wed May 30 14:32:07 GMT+02:00
   2018");

new PostDataToServer(SERVER_PATH, postDataParams).execute();

//u razredu PostDataToServer
HTTPURLConnection conn = new HTTPURLConnection();
String response = conn.send(SERVER_PATH, postDataParams);
```

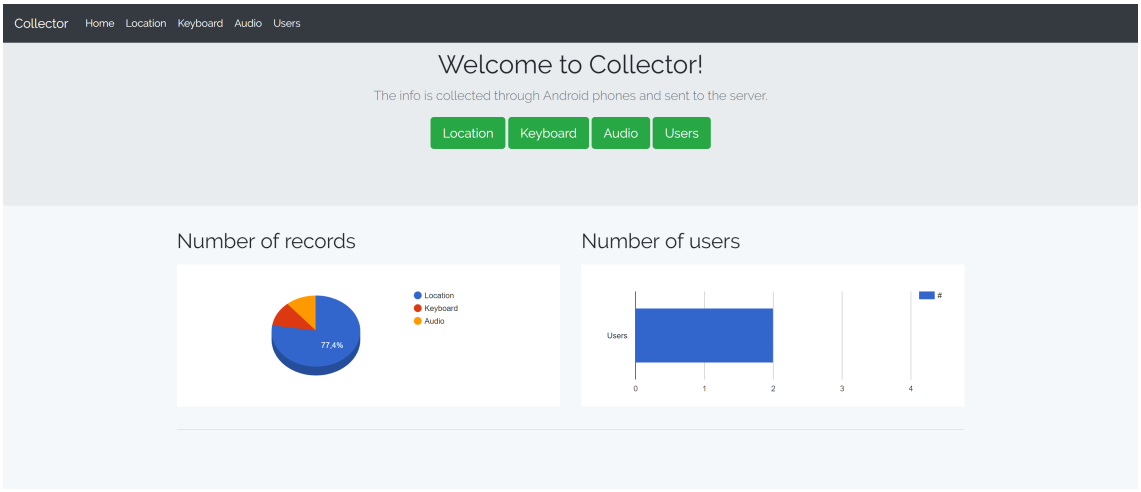
Generirani HTTP zahtjev (naravno, uz ispravne adrese servera i hosta i drugačiji boundary parametar) imati će oblik sličan sljedećem:

```
POST /serverpath HTTP/1.1
Host: example.org
Content-Type: multipart/form-data;boundary="boundary"

 --===boundary===

Content-Disposition: form-data; name="date"
Content-Type: text/plain; charset=UTF-8

Wed May 30 14:32:07 GMT+02:00 2018
 --===boundary===

Content-Disposition: form-data; name="text"
Content-Type: text/plain; charset=UTF-8

Mobilna aplikacija za prikupljanje podataka
 --===boundary===

Content-Disposition: form-data; name="account"
Content-Type: text/plain; charset=UTF-8

 baksa.mirna@gmail.com

 --===boundary===--
```

4. Web aplikacija

4.1. Uvod

Potreba za web aplikacijom javila se u svrhu pregleda podataka koji su sakupljeni od mobilnih uređaja. Za to je razvijena jednostavna CRUD (*Create, Read, Update, Delete*) web aplikacija koja detaljno prikazuje sve zapise, korisnike aplikacije te prikupljenu statistiku.



Slika 4.1: Naslovnica web aplikacije

Arhitektura

Već je bilo spomenuto da se za razvoj aplikacije koristi programski jezik PHP i radno okruženje Laravel.

Iako Laravel u suštini pruža mnogo više od relativno jednostavne MVC (*Model - View - Controller*) arhitekture, za potrebe ove aplikacije to se može zanemariti. Aplikacija će biti objašnjena kroz prizmu MVC-a, a više informacija može se dobiti u službenoj dokumentaciji Laravel radnog okruženja[1].

[1] `https://laravel.com/docs/5.6`

MVC oblikovni obrazac raslojava aplikaciju u tri dijela:

1. **modeli** - modeliranje podataka i poslovna logika.
2. **pogledi (engl. *views*)** - vizualni prikaz modeliranih podataka,
3. **upravljači (engl. *controllers*)** - upravljanje korisničkim zahtjevima.

4.2. Implementacija

S obzirom da tema rada nije izrada web aplikacije, već ona služi za pregled sakupljenih podataka, u implementaciju se neće duboko ulaziti. Implementacija je dostupna na GitHub-u [2].

Modeli

Za svaki od zapisa (lokacija, tekst, zvuk) implementiran je jedan model. Svu logiku oko modela (komunikacija s bazom podataka) Laravel vrši sam, a jedina stvar što je bilo potrebno definirati u modelima su strani ključevi na tablicu s korisnicima.

```
public function collectoruser()
   {
      return $this->belongsTo('App\CollectorUser', 'user_id');
   }
```

Upravljači

Za svaki prikupljač implementiran je po jedan Upravljač (engl. *controller*), kao i jedan za upravljanje korisnicima. U Upravljačima su implementirane funkcionalnosti čitanja zapisa iz baze podataka, spremanja zapisa, filtriranja zapisa po korisničkom računu te brisanjem zapisa. Funkcionalnost uređivanja podataka nije se pokazala kao potreba, pa nije ni implementirana.

Pogledi

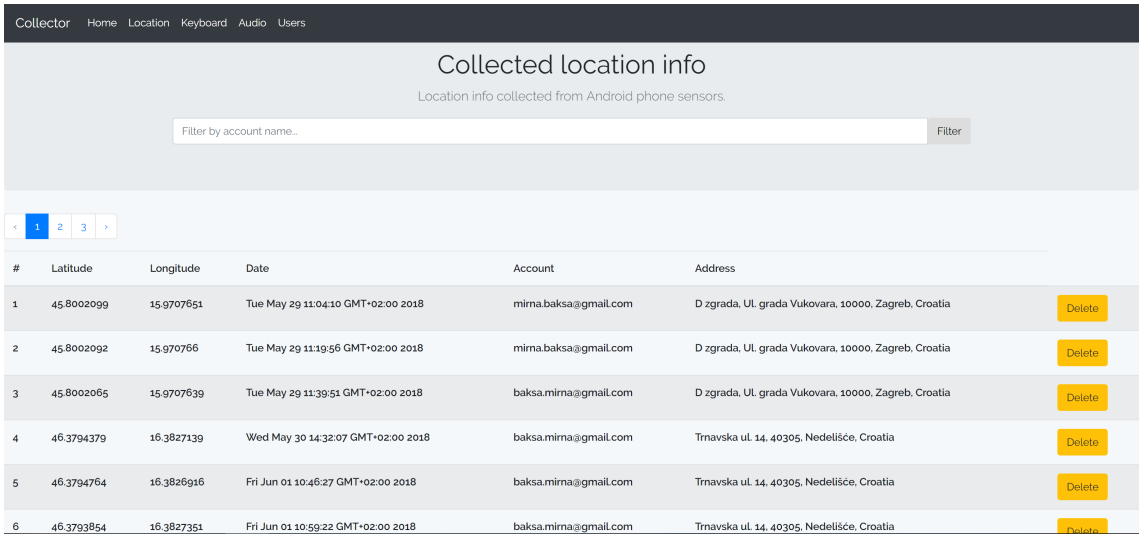
Za kreiranje pogleda (engl. *views*) Laravel koristi *blade* predloške [3]. Dizajn je rađen pomoću *Bootstrap*-a. *Bootstrap* je razvijen od strane *Twitter*-a, a nosi titulu danas

[2] https://github.com/mirnabaksa/Collector
[3] https://laravel.com/docs/5.6/blade

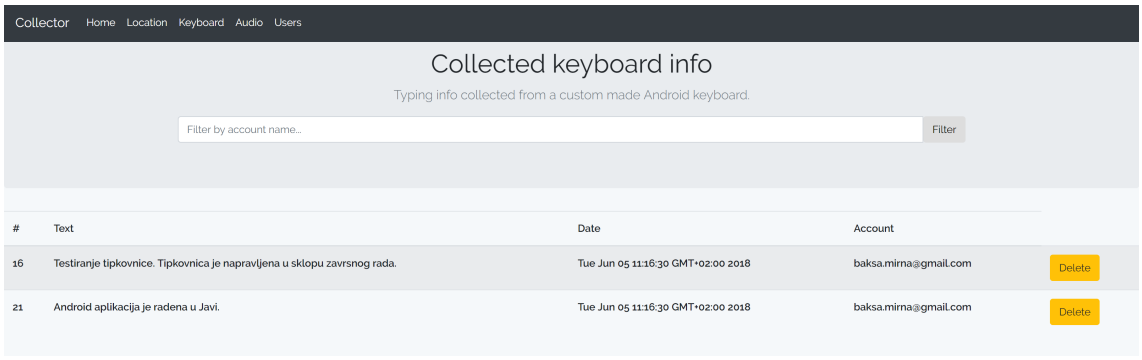
najpopularnijeg HTML, CSS i JavaScript radnog okruženja za razvoj responzivnih i konzistentnih web stranica i web aplikacija.

Uz osnovne funkcionalnosti prikaza, brisanja i filtriranja zapisa implementirana je paginacija te osnovna statistika aplikacije koja je prikazana na početnom ekranu (Slika 4.1).

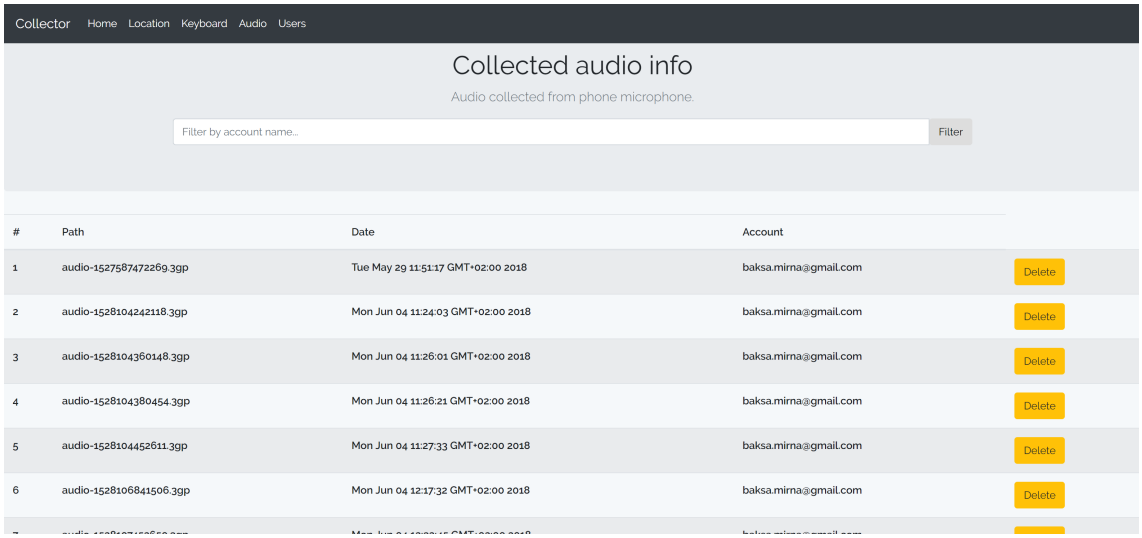
Rezultati su vidljivi na slikama 4.2, 4.3, 4.4 i 4.5. Zapisi koji su trenutno u aplikaciji su ogledni.



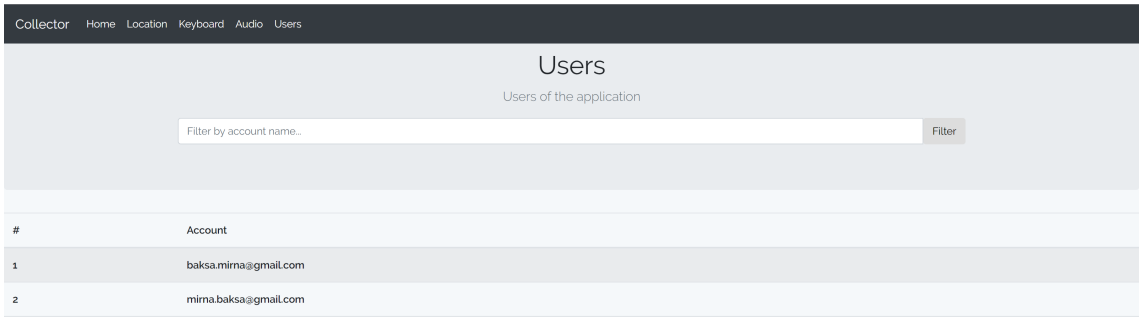
Slika 4.2: Zapisi o lokaciji



Slika 4.3: Zapisi o unesenom tekstu



Slika 4.4: Zapisi o zvuku



Slika 4.5: Zapisi o korisnicima

5. Testiranje

U sklopu ovog rada provedeno je testiranje na potrošnju baterije kod korištenja aplikacije.

S obzirom da prosječni korisnik danas koristi mobilni uređaj više-manje cijeli dan, bitno je da aplikacija ne koristi mnogo resursa uređaja. Proces dohvata podataka u razvijenoj mobilnoj aplikaciji potencijalno je velik potrošač baterije, ali testiranje je pokušalo utvrditi kolika je razlika u potrošnji baterije sa i bez pokrenute aplikacije.

Način testiranja

Testiranje je provedeno kroz dva tjedna. U šest dana svakog tjedna (ponedjeljak - subota) praćena je promjena baterije u razdoblju od 10 do 11 sati ujutro. Prvi tjedan aplikacija nije bila uključena, dok drugi tjedan je. U trenutku testiranja uređaj je bio priključen na *Wi-Fi* (a ne mobilnu mrežu). Sve druge aktivnosti na mobilnom uređaju ostale su iste kao i u svakodnevnom korištenju.

Mobitel na kojem je provedeno testiranje je *Xiaomi Redmi 4x* sljedećih specifikacija baterije:

Tablica 5.1: Specifikacije baterije

Baterija	**Li-Po 4100 mAh**
Na čekanju	do 432h (3G)
Razgovor	do 36h (3G)
Reprodukcija medija	do 87h (3G)

Izvor specifikacija je GSMArena[1]. Baterija je relativno velikog kapaciteta te u normalnoj uporabi mobilni uređaj izdrži do dva dana bez punjenja.

Rezultati testiranja dani su u tablici 5.2.

[1] `https://www.gsmarena.com/xiaomi_redmi_4_(4x)-8608.php`

Tablica 5.2: Rezultati testiranja

Status aplikacije	Promjena postotka baterije u razdoblju 10 - 11h					
	Ponedjeljak	**Utorak**	**Srijeda**	**Četvrtak**	**Petak**	**Subota**
Uključena	2	2	1	3	3	0
Isključena	3	0	2	2	3	2

Ovi su podaci bili podvrgnuti uparenom t-testu. Upareni t-test služi za usporedbu srednje vrijednosti dviju populacija u kojem opažanja iz jedne populacije mogu biti uparene s opažanjima iz druge. Testiranje koje je bilo provedeno u skladu je s ovim zahtjevom: uspoređujemo dva različita načina rada uređaja (uključena/isključena aplikacija) na istom uređaju (*Xiaomi Redmi 4x*).

Pretpostavka u korištenju ovog testa je normalna razdioba distribucije potrošnje baterije.

Testiranje je provedeno u programskom jeziku R korištenjem ugrađenih funkcija, s razinom značajnosti 5%. Hipoteze su oblikovane na sljedeći način:

H0: nema razlike u potrošnji baterije kad je aplikacija uključena i kad je isključena,

H1: ima razlike u potrošnji baterije.

```
        Paired t-test

data:  potrosnja$Ukljuceno and potrosnja$Iskljuceno
t = -0.27735, df = 5, p-value = 0.6037
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.37756      Inf
sample estimates:
mean of the differences
             -0.1666667
```

Slika 5.1: Upareni t test

Iako se već iz samih podataka moglo naslutiti da nema prevelike razlike u potrošnji baterije, ovaj test je to i potvrdio. *P* vrijednost je dovoljno velika (0.6037) da na razini značajnosti 5% možemo zaključiti da nema razlike u potrošnji baterije kad je aplikacija uključena i isključena, odnosno da ne odbijamo nultu hipotezu.

6. Zaključak

Efikasno prikupljanje podatka jedan je od najbitnijih zahtjeva današnjih aplikacija. Cilj ovog rada upravo je bio napraviti mobilnu aplikaciju koja prikuplja podatke s mobilnog uređaja.

Razvijena je mobilna aplikacija za Android operacijski sustav i prikladna web aplikacija za prikaz istih. Prvi korak bilo je oblikovanje sustava - odabir mobilnog operacijskog sustava, odabir programskog jezika i radnog okruženja web aplikacije te oblikovanje prikladnog modela. Sljedeće napravljeno bila je implementacija mobilne aplikacije u programskom jeziku Java te pronalazak prikladnog načina za slanje podataka na vanjski server, za što su odabrani jednostavni HTTP zahtjevi. Nakon razvijene mobilne aplikacije, implementirana je i CRUD web aplikacija za prikaz podataka. Na kraju je provedeno testiranje mobilne aplikacije na potrošnju baterije. Rezultati su zadovoljavajući - aplikacija je efikasna i ne troši puno resursa mobitela.

Postoji mnogo poboljšanja i proširenja razvijenog sustava koja bi pogodovala i korisnicima i onom koji skupljene podatke treba interpretirati. Neke od ideja su: kriptiranje teksta (pogoduje privatnosti korisnika), analiza zvučnih zapisa (npr. micanje praznina u zapisima), poboljšanje dizajna i dodavanje funkcionalnosti tipkovnici (npr. *autocorrect*), funkcionalnost prijave na aplikaciju kako bi se osigurala sigurnost podataka i sl.

LITERATURA

Android dokumentacija. URL `https://developer.android.com/docs/`.

Laravel dokumentacija. URL `https://laravel.com/docs/5.6/`.

Probability & Statistics for Engineers & Scientists. Pearson, 2012.

Design Patterns: Elements of Reusable Object-Oriented Software. Pearson, 2016.

POPIS SLIKA

POPIS TABLICA

Mobilna aplikacija za prikupljanje podataka o korisničkoj aktivnosti

Sažetak

Razvijena je Android aplikacija za prikupljanje podataka o lokaciji, unesenom tekstu i snimljenom zvuku s mobilnog uređaja. Podaci se preko HTTP zahtjeva šalju na server te spremaju u bazu podataka. Za pregled podataka razvijena je jednostavna web aplikacija.

Ključne riječi: Android, senzori, lokacija, tipkovnica, zvuk, HTTP, PHP

Mobile Application for Collecting User Activity Data

Abstract

An Android application was developed for collecting location, text and sound data from a mobile device. The collected data is sent to server using HTTP requests and stored in a database. A simple web application was developed for the viewing of collected data.

Keywords: Android, sensors, location, keyboard, sound, HTTP, PHP