UNIWERSYTET GDAŃSKI

WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI

Jessica Tkacz
Tomasz Wilk

Kierunek: Informatyka
Specjalność: Aplikacje internetowe i bazy danych

Aplikacja dla systemu Android usprawniająca kontakty z bliskimi.

Praca licencjacka napisana
pod kierunkiem dr Włodzimierza Bzyla

Gdańsk 2016

Spis treści

WSTĘP

Przedmiotem naszej pracy jest aplikacja napisana dla systemu Android na telefony oraz tablety. Jest ona przeznaczona dla tych, którzy cenią sobie pielęgnowanie kontaktów z rodziną i przyjaciółmi, a przy tym są zbyt zajęci bądź zapominalscy, by pamiętać o choćby cotygodniowym telefonie do danej osoby. Aplikacja służy do przypominania użytkownikowi o wykonaniu telefonu do konkretnej osoby na podstawie zaznaczonych przez użytkownika preferencji, co do częstotliwości połączeń z danym kontaktem.

Analiza dostępnych rozwiązań rozpoczęła się na długo przed podjęciem decyzji o pisaniu naszej aplikacji, jako że szukaliśmy czegoś podobnego do własnego użytku. Istnieją oczywiście aplikacje dla systemu Android, które choć po części miały spełniać podobną rolę, ale posiadały one zazwyczaj zbyt wiele skomplikowanych ustawień oraz niepotrzebnych funkcji, do których po krótkim czasie traciło się cierpliwość. Celem naszego rozwiązania jest prostota oraz brak konieczności częstego powracania do ustawień, co z kolei prowadzi do kolejnych atutów – oszczędności czasu oraz użyteczność osobom starszym, niezaznajomionym dobrze z nowymi technologiami. Ponadto aplikacja wszystkim swoim użytkownikom wysyła adnotacje z przypomnieniem o np. Dniu Dziadka albo Dniu Matki, co jest ponadprogramową okazją do wykonania telefonu.

Aplikacja ma na celu wspomóc osoby, które z różnych powodów nie pamiętają o wykonaniu telefonu do bliskich osób, choć ważne jest dla nich utrzymywanie dobrych stosunków z nimi oraz regularny kontakt. Naszym celem było by prosty w obsłudze interfejs oraz jednorazowa konieczność tworzenia ustawień sprawiły by aplikacja mogła cieszyć się popularnością wśród różnych grup wiekowych. Nasze rozwiązanie może posłużyć zarówno młodzieży, która przez nawał obowiązków nie zawsze pamięta o tym, żeby zadzwonić do ukochanej babci, jak i również osobom starszym w kontaktach z rodziną czy w przypadku regularnych wizyt u lekarza wymagających rejestracji. Ponadto aplikacja może zostać wykorzystana przez specjalistów różnych dziedzin jako wsparcie w kontakcie z klientami w regularnych odstępach czasu (np. comiesięczne przypomnienie o wizycie ortodontycznej).

2. Projekt i analiza

W tym rozdziale pragniemy przedstawić wybrane diagramy związane ze sposobem działania naszej aplikacji, jej strukturą oraz samym jej zaprojektowaniem.

UC-1: Uruchomienie aplikacji

Główny scenariusz:
1. Użytkownik otrzymuje listę kontaktów
2. Użytkownik znajduje wybrany kontakt
3. Użytkownik ustala pożądaną częstotliwość kontaktu z wybraną osobą używając paska przewijania
4. Użytkownik potwierdza wybór zaznaczając pole typu checkbox
5. Użytkownik potwierdza ustawienia przyciskiem *Zapisz*

Rozszerzenia:
5.1.

UC-2: Dodanie/Usunięcie kontaktu

Główny scenariusz:
1. Użytkownik dodaje kontakt w swojej książce adresowej(Kontakty)
2. Użytkownik znajduje dodany/edytowany kontakt na końcu listy kontaktów w aplikacji

Rozszerzenia:
1.2. Użytkownik usuwa kontakt w swojej książce adresowej
1.3. Użytkownik edytuje kontakt w swojej książce adresowej

UC-3: Ukrycie aplikacji

Główny scenariusz:
1. Użytkownik przechodzi z aplikacji do pulpitu/wychodzi z aplikacji/zostawia aplikację działającą w tle.

Rozszerzenia:
1.1.

UC-4: Naciśnięcie notyfikacji

Główny scenariusz:
1. Użytkownik rozwija pasek notyfikacji
2. Użytkownik dotyka wybraną notyfikację
3. Użytkownik przechodzi z notyfikacji do aplikacji Kontakty/Telefon z wybranym numerem

Rozszerzenia:
3.1.

2.2. Wymagania funkcjonalne i niefunkcjonalne

Wymagania funkcjonalne:

Aplikacja pobiera pełną listę kontaktów z telefonu użytkownika i wyświetla ją wraz z opcjami ustawień. Właściciel ustawia pożądaną częstotliwość przypomnień o połączeniu z danym kontaktem za pomocą *seekBar,* a obok prezentują się rezultaty wykonanych czynności w postaci ilości dni. Następnie wstępnie zapisuje te ustawienia za pomocą *checkBox*, ponieważ tylko zaznaczone w ten sposób kontakty zostaną uwzględnione przez aplikację. Jest to zabezpieczenie na wypadek, gdyby użytkownik przypadkiem ustalił priorytet w kontakcie, na temat którego powiadomień nie chce otrzymywać. Na sam koniec wystarczy potwierdzić wszystkie ustawienia za pomocą guzika na dole, co da aplikacji ostateczną wersję, którą ma wziąć pod uwagę. W każdym momencie użytkownik może powrócić do owych ustawień i je zmienić.

Wymagania niefunkcjonalne:

Oczywistym ograniczeniem jest fakt, iż aplikacja jest zorientowana na system Android i na żadnym innym nie będzie działała. Możliwe jest używanie jej zarówno na *smartfonach,* jak i tabletach posiadających wersję systemu Android 5.0. i wzwyż. Można również uruchomić aplikację w środowisku Android Studio, w którym była pisana i oglądać efekty próbnych działań na wybranym emulatorze posiadającym odpowiednią wersję systemu.

2.6 Funkcjonalności - fragmenty kodu aplikacji

a) Funkcja *setContactCalls* odpowiedzialna jest za dopasowanie połączeń do konkretnych kontaktów:

```
public void setContactCalls(Map<String, Contact> contactsMap, Map<String, ArrayList<Call>> callsMap){
final String[] numberProjection = new String[]{
        Phone.NUMBER,
        Phone.CONTACT_ID,
};

Cursor phone = new CursorLoader(context,
        Phone.CONTENT_URI,
        numberProjection,
        null,
        null,
        null).loadInBackground();

if (phone.moveToFirst()) {
    final int contactNumberColumnIndex = phone.getColumnIndex(Phone.NUMBER);
    final int contactIdColumnIndex = phone.getColumnIndex(Phone.CONTACT_ID);

    while (!phone.isAfterLast()) {
        final String numberBeforeConversion = phone.getString(contactNumberColum
        final String contactId = phone.getString(contactIdColumnIndex);
        String number = numberBeforeConversion.replaceAll("\\D+","");

        Contact contact = contactsMap.get(contactId);
        ArrayList<Call> calls = callsMap.get(number);

        Call recentCall = new Call();
        if(calls != null) {
            for (Call c : calls){
                if(Integer.parseInt(c.getDuration()) > 0){
                    if(c.getDate().compareTo(recentCall.getDate())>0){
                        recentCall = c;
                    }
                }
            }
        }

        if (contact == null) {
            continue;
        }

        contact.setRecentCall(recentCall);

        Log.i("TUTAJ numer+data", number +" --> "+ contact.recentCall.getDate());
        phone.moveToNext();
    }
}

phone.close();
```

b) Funkcja *CallNotification* odpowiedzialna za pokazanie pojedynczej notyfikacji:

```
public class CallNotification {

    static int notificationNumber = 0;
    public CallNotification(){
    }

    public void showNotification(String message, String tittle, Context context, int id, String phoneNumber) {
        Intent intent = new Intent(Intent.ACTION_DIAL);
        intent.setData(Uri.parse("tel:"+phoneNumber));

        PendingIntent pi = PendingIntent.getActivity(context, 0, intent, 0);

        Notification notification = new NotificationCompat.Builder(context)
                .setSmallIcon(R.drawable.babcia)
                .setContentTitle(tittle)
                .setContentText(message)
                .setContentIntent(pi)
                .setAutoCancel(true)
                .build();

        NotificationManager notificationManager = (NotificationManager) context.getSystemService(context.NOTIFICATION_SERVICE);
        notificationManager.notify(id, notification);
        notificationNumber++;
    }
```

c) Głowa programu odpowiedzialna za wybieranie kontaktów nadających się do wysłania notyfikacji:

```
if(c.isChecked()){
    Log.i("longCall", c.name + " " + c.delay + " od ostatniej rozmowy "+ (c.getProgression()*(1000)));
    String tittle = "zadzwon do " + c.getName();
    String message = "Dzwoniles dawniej niz "+ c.getProgression() + " minut temu";
    String number = "" ;
    if (c.numbers.size() > 0 && c.numbers.get(0) != null) {
        number = c.numbers.get(0).number;
    }

    if(c.getDelay() > c.getProgression()*(1000*60)){
        Log.i("longCall", message);
        new AlarmBroadcaster(context,tittle , message, number, c.id).setAlarmBroadcast(0);
    } else {
        Log.i("longCall", "dzwoniles do " + c.name + " zadzwon za " + (c.getProgression()-(c.getDelay()/(1000*60))));
        new AlarmBroadcaster(context, tittle, message,number, c.id)
                .setAlarmBroadcast((int) (c.getProgression()-(c.getDelay()/(1000*60))));
    }

} else {
    new AlarmBroadcaster(context).cancelAlarm(c.id);

}
```

d) Klasa odpowiedzialna za tworzenie alarmów systemowych, które po danym czasie wywołają funkcje wyświetlającą notyfikacje. + możliwość anulowania zakolejkowanych notyfikacji:

```java
public class AlarmBroadcaster {
    private Context context;
    private PendingIntent reminderBroadcastIntent;

    public  AlarmBroadcaster(Context context) { this.context = context; }
    public AlarmBroadcaster(Context context, String tittle, String message, String phoneNumber, String id) {
        this.context = context;
        Intent intent = new Intent(context, AlarmReceiver.class);
        intent.putExtra("tittle", tittle);
        intent.putExtra("message", message);
        intent.putExtra("id", id);
        intent.putExtra("phoneNumber", phoneNumber);
        reminderBroadcastIntent = PendingIntent.getBroadcast(context, Integer.parseInt(id), intent, PendingIntent.FLAG_UPDATE_CURRENT);

    }

    public void setAlarmBroadcast(int minutes){
        //Set the alarm to 10 seconds from now
        Calendar c = Calendar.getInstance();
        c.add(Calendar.MINUTE,  minutes);
        long when = c.getTimeInMillis();
        // Schedule the alarm!
        AlarmManager alarmToBroadcast = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
        alarmToBroadcast.set(AlarmManager.RTC_WAKEUP, when, reminderBroadcastIntent);
    }
    public void cancelAlarm(String id){
        AlarmManager alarmToCancel = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
        Intent intent = new Intent(context, AlarmReceiver.class);
        PendingIntent pendingIntentToCancel = PendingIntent.getBroadcast(context, Integer.parseInt(id), intent, PendingIntent.FLAG_UPDATE_CURRENT);
        alarmToCancel.cancel(pendingIntentToCancel);

    }
}
}
```

e) *ViewDisplay* jest adapterem, który zajmuje się wyświetlaniem zapisanych ustawień użytkownika, lub (jeżeli nie ma zapisanych) wyświetla domyślne ustawienia.

```
public class ViewDisplay extends ArrayAdapter<Contact>{


    public ViewDisplay(Context context, ArrayList<Contact> contacts) { super(context, 0, contacts); }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item
        Contact contact = getItem(position);
        // Check if an existing view is being reused, otherwise inflate the view
        View view = convertView;
        if (view == null) {
            LayoutInflater inflater = LayoutInflater.from(getContext());
            view = inflater.inflate(R.layout.single_contact, parent, false);
        }
        // Populate the data into the template view using the data object
        TextView name = (TextView) view.findViewById(R.id.tvName);
        TextView phone = (TextView) view.findViewById(R.id.tvPhone);
        SeekBar contactSeekBar = (SeekBar) view.findViewById(R.id.contactSeekBar);
        TextView seekBarProgress = (TextView) view.findViewById(R.id.seekBarProgress);
        CheckBox checkBox = (CheckBox) view.findViewById(R.id.remindCheckBox);

        contact.setSeekBar(contactSeekBar);
        contact.setProgressValue(seekBarProgress);
        contact.setCheckBox(checkBox);
        contact.initializeContact();
        contactSeekBar.setProgress(contact.getProgression());
        seekBarProgress.setText(""+contact.getProgression());
        checkBox.setChecked(contact.isChecked());



        name.setText(contact.name);
        phone.setText("");

        if (contact.numbers.size() > 0 && contact.numbers.get(0) != null) {
            phone.setText(contact.numbers.get(0).number);
        }

        return view;
    }
}
```

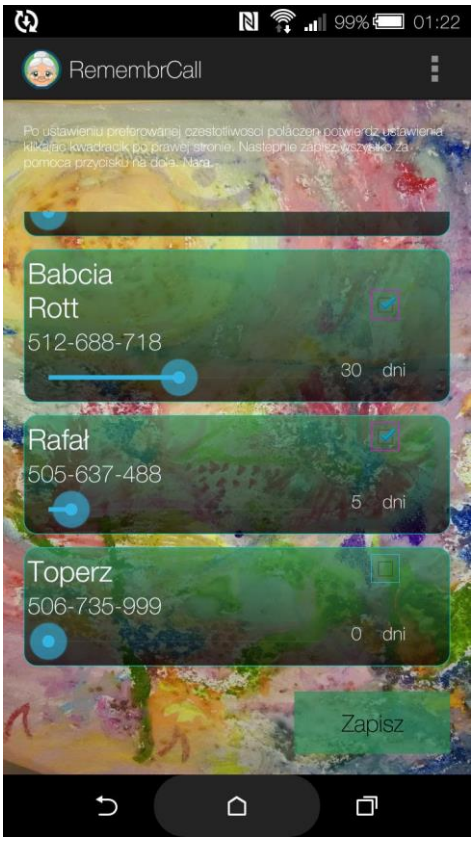
g) Funkcja tworząca mapę połączeń przypisanych do numeru na podstawie historii połączeń telefonu.

```
public void getCallDetails(Context context) {
    Log.i("LogDetails12", "getCallDetails");

    final String[] numberProjection = new String[]{
            CallLog.Calls.NUMBER,
            CallLog.Calls.DATE,
            CallLog.Calls.DURATION,
            CallLog.Calls.CACHED_NAME,
    };

    Cursor callDetailsCursor = new CursorLoader(context,
            CallLog.Calls.CONTENT_URI,
            numberProjection,
            null,
            null,
            null).loadInBackground();

    int number = callDetailsCursor.getColumnIndex(CallLog.Calls.NUMBER);
    int date = callDetailsCursor.getColumnIndex(CallLog.Calls.DATE);
    int duration = callDetailsCursor.getColumnIndex(CallLog.Calls.DURATION);
    int name = callDetailsCursor.getColumnIndex(CallLog.Calls.CACHED_NAME);


    while (callDetailsCursor.moveToNext()) {
        String phNumber = callDetailsCursor.getString(number);
        String callDate = callDetailsCursor.getString(date);
        String contactName = callDetailsCursor.getString(name);
        Date callDateTime = new Date(Long.valueOf(callDate));
        String callDuration = callDetailsCursor.getString(duration);


        if(callsMap.get(phNumber) != null)
        {
            ArrayList<Call> calls = callsMap.get(phNumber);
            calls.add(new Call(callDateTime, callDuration, contactName));
            callsMap.put(phNumber, calls);

        } else {
            ArrayList<Call> calls = new ArrayList<>();
            calls.add(new Call(callDateTime, callDuration, contactName));
            callsMap.put(phNumber, calls);
        }

    }
    callDetailsCursor.close();

}
```

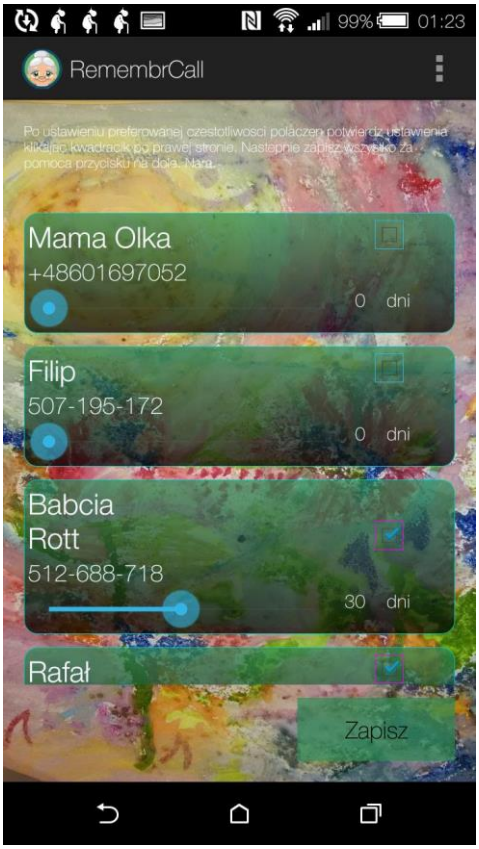
3.2 Użyte technologie

Aplikacja została napisana w języku programowania Java w środowisku Android Studio w wersji 2.1.. Aplikacja zaprogramowana została z myślą o użytkownikach systemu Android 5.0. bądź późniejszym. Program kompilowany był dla wersji Android API 21., a buildowany dokładnie dla wersji 21.1.2. poprzez *gradle* 2.1. Odbiór zewnętrznych wiadomości został zrealizowany przez GCM *(*ang. *Google Cloud Messages)* przy pomocy *pushbots* w wersji 2.0.13.. Front-end występuje w postaci plików xml (ang. *Extensible Markup Language*) w wersji 1.0. z kodowaniem UTF-8.
Projekt interfejsu użytkownika.

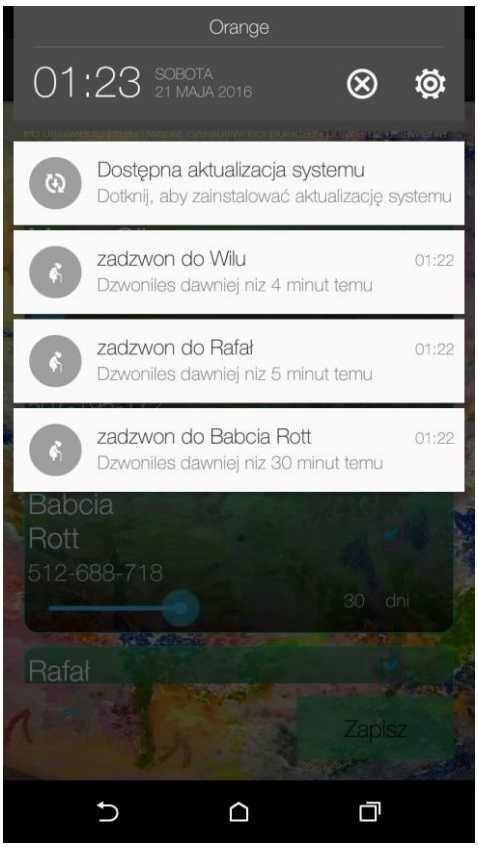
Strona główna po uruchomieniu aplikacji RemembrCall. Wyświetlają się tutaj wszystkie kontakty pobrane z telefonu. Do każdego kontaktu widzianego w osobnej ramce mamy opcje wyboru częstotliwości połączeń oraz pole do zaznaczenia aby ustawienia zostały wzięte pod uwagę. Na dole po prawej stronie widnieje przycisk „Zapisz”, którego wciśnięcie jest konieczne do działania aplikacji. Na samej górze znajduje się krótki opis korzystania z niej.



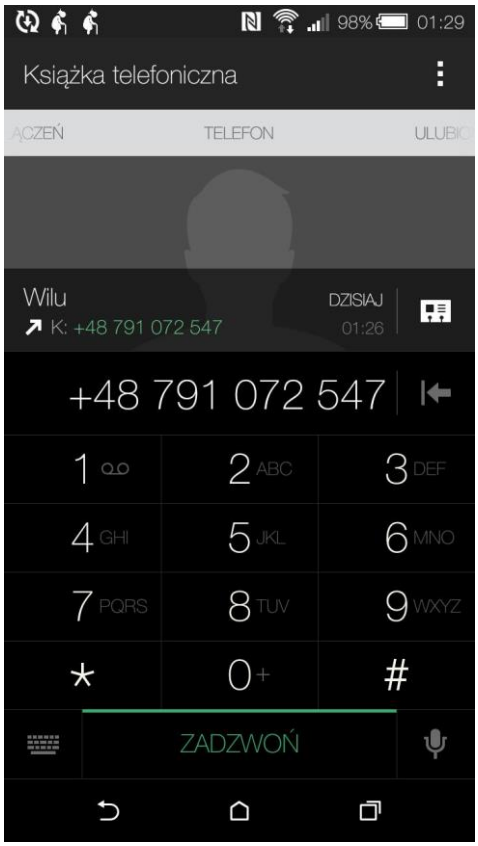
Przykładowy widok notyfikacji przypominających o wykonaniu telefonu do konkretnej osoby, wysłanych po wybranym przez użytkownika czasie. Adnotacja przybiera kształt spacerującej o lasce staruszki, co nawiązuje tematycznie do ikony całej aplikacji.



Po rozwinięciu czarnego paska u góry pokazują się szczegóły poszczególnych notyfikacji takie jak nazwa kontaktu oraz ilość dni, które minęły od ostatniego połączenia z nim.



Po kliknięciu konkretnej adnotacji aplikacja przekierowuje użytkownika bezpośrednio do Książki telefonicznej z już wybranym numerem kontaktu, odnośnie którego notyfikację wybraliśmy. Jest to o tyle komfortowe, że dopóki nie klikniemy adnotacji, będzie nam ona wciąż przypominała o tym, żeby zadzwonić, a gdy zostanie już wybrana poniekąd zmusi użytkownika do wykonania zaplanowanego telefonu.



3. Implementacja

3.3 Testowanie aplikacji

Testowanie aplikacji przeprowadzono manualnie wykonując wybrane operacje w aplikacji.

SŁOWA KLUCZOWE:
„aplikacja” „Android” ”rodzina” „przypomnienia” „prostota” „adnotacje” „notyfikacje” „kontakt” „organizacja” „oszczędność czasu” „zadzwonić” „telefon” „systematyczność” „regularny” „połączenie”