SMS Scheduling App

Student,

Ionescu Tiberiu Dan,

An IV, CTI

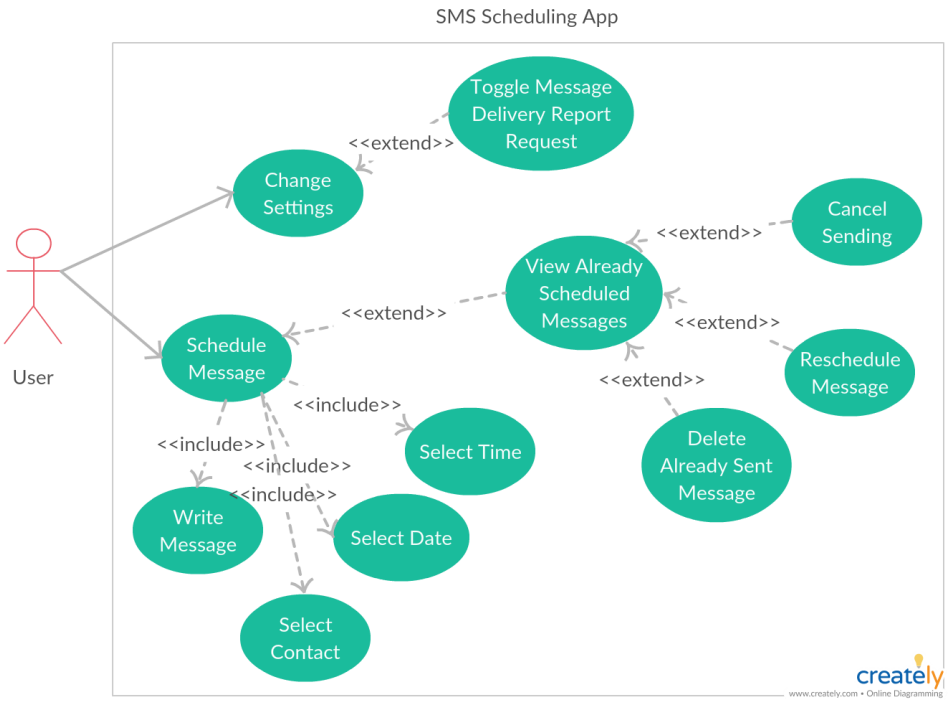
1. INTRODUCTION

Sometimes the need for making things happen in the future arises, so my idea for the project was to make a mobile app which would allow the user to schedule a SMS message to be sent at a specific time.

The reason behind my choice for the topic of the project is because I find it useful and it could make things easier for you if, for example, you keep forgetting to send a birthday SMS message to someone, or if you have that one friend who doesn't seem to know what an alarm is, or in case you need to send reminders, maybe you need to send periodically SMS for parking and so on.

2. DESIGN AND IMPLEMENTION

Below is the use case diagram for the app:



Schedule messages: this is the main use case of the app. It requires the user to provide a date and time for the message to be sent, the recipient number for the message, which can be chosen from the phone contacts, and the text message.

In order to send a message to the moment chosen by the user, I used the *AlarmManager* class to access to the system alarm services and *WakefulBroadcastReceiver* and *BroadcastReceiver* class to receive and handle broadcasted intents.

After the message have been scheduled, it is inserted into a database. For the database I used SQLite. Unless they are deleted by the user, all scheduled messages can be seen on the first page of the app along with their status ("Sent", "Failed", "Will be sent"). From there you can tap on them in order to reschedule the date, change the message content or cancel the sending entirely. Once a message is sent, the user will receive a notification to tell him if the message was sent successfully or not.

Access settings menu: in the right corner of the menu bar there are a “three dots” symbol representing the menu from where you can choose to receive or not delivery reports for your sent messages. This is an optional setting because some phone carriers are taxing for delivery reports. It is disabled by default.

Implementation:

In this section I will explain how I implemented the main functionalities of the application and I will provide some code examples for the implementations.

```
public class SmsModel {

    public static final String ERROR_UNKNOWN = "UNKNOWN";
    public static final String ERROR_GENERIC = "GENERIC";
    public static final String ERROR_NO_SERVICE = "NO_SERVICE";
    public static final String ERROR_NULL_PDU = "NULL_PDU";
    public static final String ERROR_RADIO_OFF = "RADIO_OFF";

    public static final String STATUS_PENDING = "PENDING";
    public static final String STATUS_SENT = "SENT";
    public static final String STATUS_DELIVERED = "DELIVERED";
    public static final String STATUS_FAILED = "FAILED";

    private long timestampCreated;
    private long timestampScheduled;
    private String recipientNumber;
    private String recipientName;
    private String message;
    private String status = STATUS_PENDING;

    private String result = "";
```

The *SmsModel* class is used as a data structure to group all relevant information, from application’s point of view, about scheduled messages. The fields which begin with “ERROR_” are used as flags in order to track different kind of errors that are possible to be encountered during message sending. “STATUS_” fields represent the possible status in which each message may be, *timestampCreated* is used as an ID in the database, *timestampScheduled* represents the time for message to be sent.

```
public void scheduleSms(View view) {
    DatePicker formDate = (DatePicker) findViewById(R.id.form_date);
    timeScheduled.set(GregorianCalendar.YEAR, formDate.getYear());
    timeScheduled.set(GregorianCalendar.MONTH, formDate.getMonth());
    timeScheduled.set(GregorianCalendar.DAY_OF_MONTH, formDate.getDayOfMonth());
    if (timeScheduled.getTimeInMillis() < GregorianCalendar.getInstance().getTimeInMillis()) {
        Toast.makeText(getApplicationContext(), "Please choose future date", Toast.LENGTH_SHORT).show();
        return;
    }
    sms.setTimestampScheduled(timeScheduled.getTimeInMillis());

    EditText formMessage = (EditText) findViewById(R.id.form_input_message);
    sms.setMessage(formMessage.getText().toString());

    sms.setStatus(SmsModel.STATUS_PENDING);

    DbHelper.getDbHelper(this).save(sms);

    scheduleAlarm(sms);

    Intent returnIntent = new Intent();
    setResult(RESULT_SCHEDULED, returnIntent);
    finish();
}
```

The picture above shows the function for SMS scheduling. In method's first part I read the information given by user from the views and save them in the *sms* field of the *AddSmsActivity*. After that, the *sms* object is saved in the database and an alarm is scheduled.

```
private void scheduleAlarm(SmsModel sms) {
    AlarmManager alarmMgr = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        alarmMgr.setExact(AlarmManager.RTC_WAKEUP, sms.getTimestampScheduled(), getAlarmPendingIntent(sms));
    }else{
        alarmMgr.set(AlarmManager.RTC_WAKEUP, sms.getTimestampScheduled(), getAlarmPendingIntent(sms));
    }
}
```

Here is the implementation for the alarm scheduling method. As I said earlier, I used an *AlarmManager* object for doing so. Beginning in API 19, the trigger time passed to set method is treated as inexact and the alarm will not be delivered before this time, but may be deferred and delivered some time later, so if the API is 19 or later, I use *setExact* method in order to avoid this problem.

```
public void unscheduleSms(View view) {
    DbHelper.getDbHelper(this).delete(sms.getTimestampCreated());

    unscheduleAlarm(sms);

    Intent returnIntent = new Intent();
    setResult(RESULT_UNSCHEDULED, returnIntent);
    finish();
}
```

To unschedule a message, I cancel the alarm which was previously launched for that specific message calling *unscheduleAlarm* method and I remove the message from the database.

One of the problems which I encountered was to figure a way to relaunch all alarms for the pending messages after the app was rebooted. To resolve this problem, I used the class *BootReceiver* which extends *BroadcastReceiver*.

The methods below are all members of *BootReceiver* class. In *onReceive* method we get all pending messages from the database, after we check if the app was rebooted, then we call *scheduleAlarm* for each one. *ScheduleAlarm* method works *similarily* with the method with the same name mentioned above, except we do not create new SMS messages to add to the database, but we read from it already existing messages, after we check that message status is "STATUS_PENDING". This status showing that the message is yet to be sent.

```
@Override
public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)) {
        this.context = context;
        ArrayList<SmsModel> pendingSms = getPendingSms();
        Iterator<SmsModel> i = pendingSms.iterator();
        SmsModel sms;
        while (i.hasNext()) {
            sms = i.next();
            scheduleAlarm(sms);
        }
    }
}

private ArrayList<SmsModel> getPendingSms() {
    return DbHelper.getDbHelper(context).get(SmsModel.STATUS_PENDING);
}

private void scheduleAlarm(SmsModel sms) {
    AlarmManager alarmMgr = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    Intent intent = new Intent(AlarmReceiver.INTENT_FILTER);
    intent.putExtra(DbHelper.COLUMN_TIMESTAMP_CREATED, sms.getTimestampCreated());
    PendingIntent alarmIntent = PendingIntent.getBroadcast(context, sms.getId(), intent,
            flags: PendingIntent.FLAG_UPDATE_CURRENT & Intent.FILL_IN_DATA);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        alarmMgr.setExact(AlarmManager.RTC_WAKEUP, sms.getTimestampScheduled(), alarmIntent);
    }else{
        alarmMgr.set(AlarmManager.RTC_WAKEUP, sms.getTimestampScheduled(), alarmIntent);
    }
}
```

DbHelper class contains all the methods used to edit and update the database. For example, *public void save(SmsModel sms)* method is used for saving scheduled messages into the database, *public void delete(Long timestampCreated)* method is used to delete messages from database, providing a *timeStamp* as an ID in order to identify the message, *public SmsModel get(long timestampCreated)* to retrieve a *SmsModel* object from the database.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_SMS_TABLE = "CREATE TABLE " + TABLE_SMS +
            "(" +
            COLUMN_TIMESTAMP_CREATED + " BIGINTEGER PRIMARY KEY," +
            COLUMN_TIMESTAMP_SCHEDULED + " BIGINTEGER," +
            COLUMN_RECIPIENT_NUMBER + " TEXT," +
            COLUMN_RECIPIENT_NAME + " TEXT," +
            COLUMN_MESSAGE + " TEXT," +
            COLUMN_STATUS + " TEXT," +
            COLUMN_RESULT + " TEXT" +
            ")";
    db.execSQL(CREATE_SMS_TABLE);
}
```

Database Structure

For the case when message delivery reports are requested by the user, I used *SmsDeliveredReceiver* as the *BroadcastReceiver* to listen for *SmsDeliveredService* as the intent. Below are the overridden *onReceive* method from the receiver class which shows what happens when the intent is received and *onHandleIntent* method from the intent class used.

```
@Override
public void onReceive(Context context, Intent intent) {
    Intent service = new Intent(context, SmsDeliveredService.class);
    service.putExtras(intent.getExtras());
    service.putExtra(RESULT_CODE, getResultCode());
    startWakefulService(context, service);
}
```

```
@Override
protected void onHandleIntent(Intent intent) {
    long smsId = intent.getExtras().getLong(DbHelper.COLUMN_TIMESTAMP_CREATED,  defaultValue: 0);
    if (smsId == 0) {
        throw new RuntimeException("No SMS id provided with intent");
    }
    Context context = getApplicationContext();
    SmsModel sms = DbHelper.getDbHelper(context).get(smsId);
    sms.setStatus(SmsModel.STATUS_DELIVERED);
    DbHelper.getDbHelper(context).save(sms);
}
```

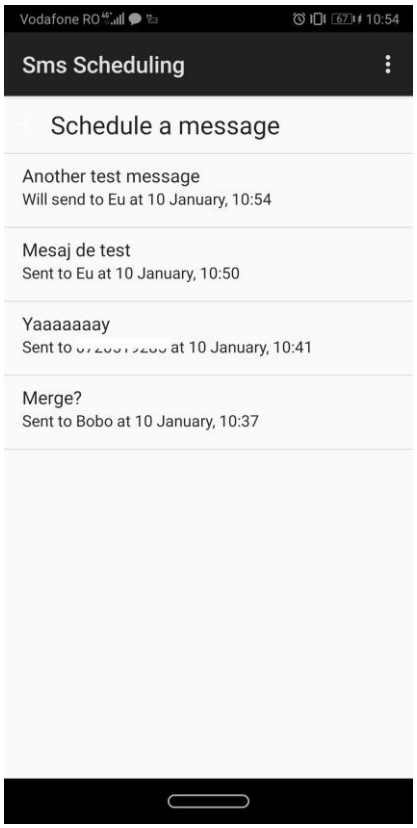
3. STATE OF THE ART

Two of the most successful apps for message scheduling are “Do It Later”, with over 100 000 downloads and “Phone Schedule” with 50 000+ downloads. Both are much more complex apps than the one I created, having support not only for message scheduling but also for email sending, phone calling or posting on social media.

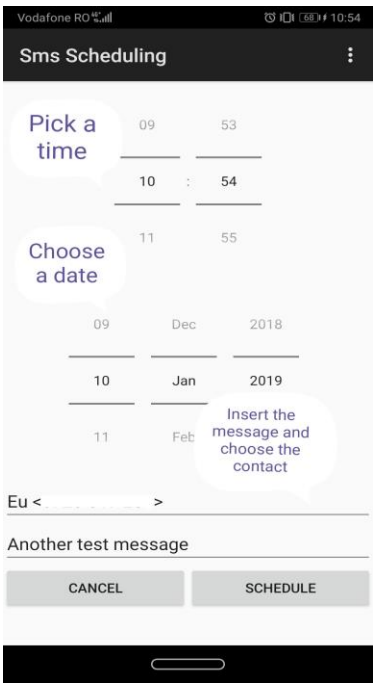
Some of the pros these apps have in comparison to mine, in terms of the message scheduling part, are:

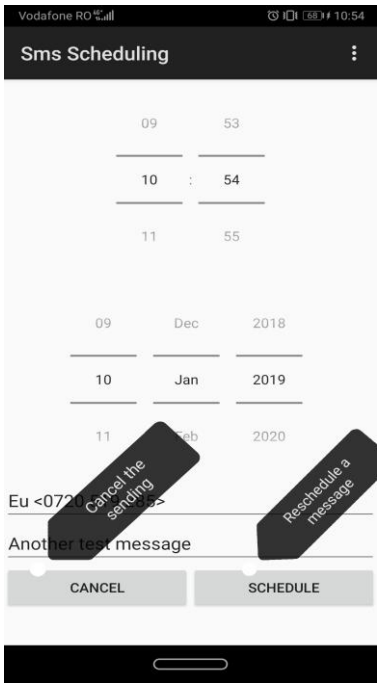
- User can choose multiple options for delay frequency (“Weekly”, “Daily”, “Hourly”)
- Predefined templates for text messages
- Dual SIM support
- Input message using speech recognition
- Various settings to personalize
- Support for multiple languages

4. Usage

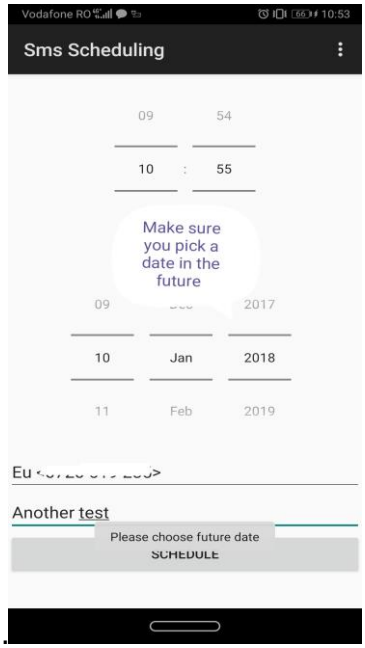


This is the Main Activity of the app. Here we have the menu bar which contains app's name "Sms Scheduling" in the left corner and settings button in the right one. Below the menu bar we have "Schedule a message" button which does exactly what it says. Under the button the app shows you all messages that were scheduled, displaying also their status, recipient, date and hour at which will be send or were sent.

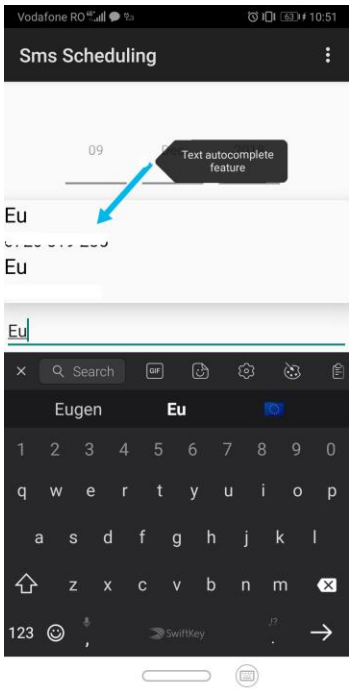




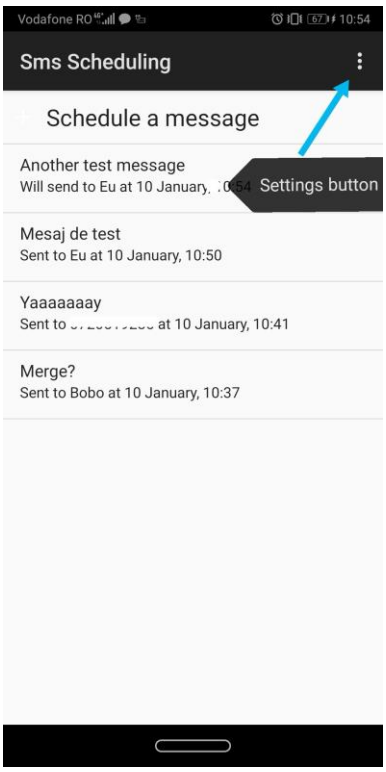
Tapping onto an already scheduled message will take you to the edit message screen from where you can easily edit any aspect of your soon to be sent message

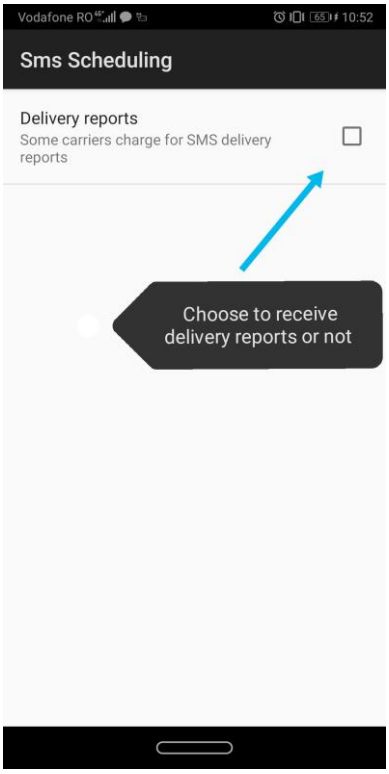


Make sure that you choose a date and time in the future for your message to be sent or else a warning will be shown.

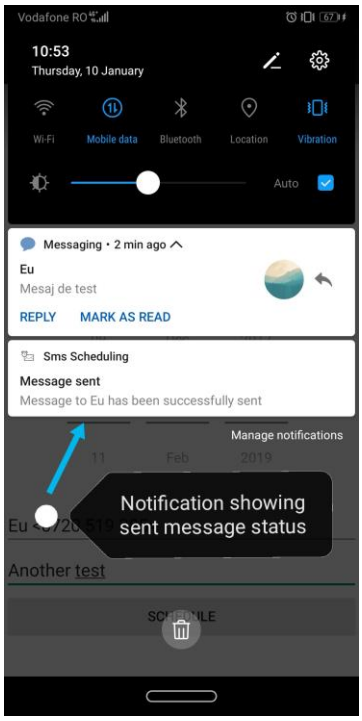


When choosing a contact to send the message to, the app will show you all available persons in your contact book beginning with the name or number given.





The settings button will take you to the settings menu from where you can choose to receive delivery reports or not.



The app will notify you if your message was sent successfully as planned or if some error occurred and it failed to send the message.

5. CONCLUSIONS

In the near future, I planned work a little bit more on the interface and design of this app, for example I would like to keep each message sorted in lists, ordered by their status (“Will send", “Sent", “Failed”) and make it more “user friendly” by adding support for more languages and changing the layout. Also, I would like to add more features to it like being able to choose the frequency for the message to be sent (“Hourly”, “Daily”), add some predefined message templates from which the user could choose and maybe integrate some mail scheduling features.