AutotasX

Iteration 2 code

Database helper class

This class handles the creation of database tables during application load and also checks if there is an upgrade in the database version.

```
public class DatabaseHelper extends SQLiteOpenHelper {

   private static final String TAG = DatabaseHelper.class.getName();

   // -- Parameters for Database creation
   private static final String DATABASE_NAME = "autotasx.DB";
   private static final int DATABASE_VERSION = 1;
   //DatabaseHelper dbHelper;
   ///private SQLiteDatabase database;

   public DatabaseHelper(Context context)
   {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }
   //Database create SQL command for LOCATION table
   private static final String LOCATION_CREATE = "CREATE TABLE LOCATION "
         + "(_id INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT, "
         + " LATITUDE REAL, LONGITUDE REAL, "
         + " RADIUS REAL , " + " SMS INTEGER DEFAULT 0, " + " WIFI INTEGER
DEFAULT 0, " + " SILENT INTEGER DEFAULT 0, " + " REMINDER INTEGER
DEFAULT 0, " + " MESSAGE TEXT NULL);";
   @Override
   public void onCreate(SQLiteDatabase database)
   {
      Log.i(TAG, "Creating databases");
      database.execSQL(LOCATION_CREATE);
   }

  //Upgrade Database on version change
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
   {
      Log.w(TAG, "Upgrading database from version " + oldVersion + " to " +
newVersion
            + ", which will destroy all old data");
      db.execSQL("DROP TABLE IF EXISTS " + LOCATION_CREATE);
      onCreate(db);
   }

}
```

ManageDB class

This class contains methods to SQLiteDatabase DML queries like select, insert and updates.

```
public class ManageDB
{

   private static final String TAG = ManageDB.class.getSimpleName();

   // -- Database objects
   //SharedPreferences prefs;
   private SQLiteDatabase database;
   private DatabaseHelper dbHelper;



   public ManageDB(Context context)
   {
      dbHelper = new DatabaseHelper(context);
   }

   public void open() throws SQLException
   {
      database = dbHelper.getWritableDatabase();
   }

   public void close()
   {
      dbHelper.close();
   }


   public void addLocation(GeoFence geoFence)
   {
      ContentValues values = new ContentValues();
      values.put("NAME", geoFence.getNameLoc());
      values.put("LATITUDE", 0);
      values.put("LONGITUDE", 0);
      values.put("RADIUS", 0);
      values.put("SMS",ActionFragment.smsVar);
      values.put("WIFI",ActionFragment.wifiVar);
      values.put("SILENT", ActionFragment.silVar);
      values.put("REMINDER",ActionFragment.remVar);
      values.put("MESSAGE",geoFence.getRemmsg());
```

```
        long insertId = database.insert("LOCATION", null, values);

        //Log.i(TAG, "Exiting createProfile(), id after row insertion - " + insertId);
        //Log.i(TAG, "ActionFregamnet sms " + ActionFragment.smsVar);
    }


    //Print a particular row from a Database
    public Cursor getEntry(GeoFence geoFence) {
       Cursor tmpCursor = database.query("LOCATION", new String[] {"_id",
"NAME","LATITUDE","LONGITUDE", "RADIUS", "SMS", "WIFI","SILENT",
"REMINDER","MESSAGE" }, "NAME" + " = '" + geoFence.getNameLoc() + "'" ,null,
null, null,null, null);
        tmpCursor.moveToFirst();
        return tmpCursor;
    }

    //Update a particular row from a Database
    public boolean updateEntry(GeoFence geoFence) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("LATITUDE", geoFence.getPoint().latitude);
        contentValues.put("LONGITUDE", geoFence.getPoint().longitude);
        contentValues.put("RADIUS", geoFence.getRadius());
        return database.update("LOCATION", contentValues, "NAME" + " = '" +
geoFence.getNameLoc() + "'",    null) > 0;
    }

}
```

Floating Action Button: Material Design

This method handles the Floating Action Button actions and animations. This is a part of the Google Material Design Specification.

```
public FloatingActionButton(Context context, AttributeSet attrs, int defStyleAttr,
                            int defStyleRes) {
        super(context, attrs, defStyleAttr);
        setClickable(true);
        // Set the outline provider for this view. The provider is given the outline which it
can
        // then modify as needed. In this case we set the outline to be an oval fitting the
height
        // and width.
        setOutlineProvider(new ViewOutlineProvider() {
            @Override
```

```
        public void getOutline(View view, Outline outline) {
            outline.setOval(0, 0, getWidth(), getHeight());
        }
    });
    // Finally, enable clipping to the outline, using the provider we set above
    setClipToOutline(true);
}

/**
 * Sets the checked/unchecked state of the FAB.
 * @param checked
 */
public void setChecked(boolean checked) {
    // If trying to set the current state, ignore.
    if (checked == mChecked) {
        return;
    }
    mChecked = checked;
    // Now refresh the drawable state (so the icon changes)
    refreshDrawableState();

    if (mOnCheckedChangeListener != null) {
        mOnCheckedChangeListener.onCheckedChanged(this, checked);
    }
}
```