Лабораторная работа №15. Работа с настройками и БД

Задание: создать приложение, работающее с SharedPreferences и сохраняющее настройки, а также работающее с БД SQLite - заполняющее БД по нажатию кнопки 1 с помощью EditText, и выводящее все записи этой БД в какой-нибудь интерфейсный элемент с помощью кнопки 2 (в виде списка, datagrid или просто правильно настроенного TextView).

Практически во всех приложениях есть какие-то данные, которые надо сохранять при перезапуске приложения, например настройки. Чтобы приложение имело возможность сохранить данные, обычно используют Shared Preferences.

Создайте проект с Empty Views Activity, и когда он построится, добавьте в него Settings Views Activity: правой кнопкой по папке app -> New -> Activity -> Settings Views Activity; и затем перейдите в файл SettingsActivity.java / SettingsActivity.kt. В последних версиях Android Studio в этом файле используется уже готовый фрагмент с некоторыми настройками, поэтому менять его не надо, в открывшемся окне просто нажмите кнопку Finish.

Добавим в MainActivity.java / MainActivity.kt опциональное меню с пунктом о программе, который открывает диалоговое окно с вашими ФИО и краткой информацией о приложении. Для этого сначала добавим новую папку menu в папку ресурсов res, и в ней создадим файл menu.xml. Откроем этот файл. Меню теперь можно добавлять как кодом, например такой строкой в коде (вкладка Code) файла menu.xml:

```
<item android:id="@+id/settings" android:title="Настройки"/>
```

Но гораздо удобнее это делать с помощью визуального взаимодействия (вкладка Design) с интерфейсом: в Palette уже видны все необходимые элементы, выберем Menu Item и перетащим его в интерфейс. Откроются свойства этого пункта меню, которые можно отредактировать, например id установить в settings, а title установить в значение Settings. Таким же образом можно добавить второй пункт опционального меню About, рассказывающий пользователю о разработчике приложения, версии приложения и т.д.

Далее добавим в файл MainActivity.java / MainActivity.kt методы onCreateOptionsMenu и onOptionsItemSelected. В код метода onOptionsItemSelected добавим обработку нажатия пункта меню Settings:

Java:

```
if (item.getItemId()==R.id.settings) {
   Intent intent = new Intent(MainActivity.this, SettingsActivity.class);
   startActivity(intent);
}
```

Kotlin:

```
if (item.itemId==R.id.settings) {
    val intent = Intent(this@MainActivity,SettingsActivity::class.java)
    startActivity(intent)
}
```

Запустите приложение, откроется пустое окно, содержащее только опциональное меню, см. рис. 52. Но… нет, никакого опционального меню нет! Чтобы оно появилось, надо либо добавить toolbar в интерфейс приложения activity_main.xml, либо в коде создать action bar / tool bar, либо просто поменять тему в папке values -> themes, потому что по умолчанию там стоит тема с названием NoActionBar:

```
<style name="Base.Theme.MyApp15KO" parent="Theme.Material3.DayNight.NoActionBar">
```

Просто уберите .NoActionBar в названии темы-родителя, и всё заработает! Запустите приложение, перейдите в настройки, измените параметры настроек и выйдите из приложения. Затем снова запустите приложение и вернитесь в настройки, параметры остались такими же, какими они были до выхода из приложения. Пример стандартных настроек приведён на рисунке 53.

Возможно, у вас не будет работать кнопка Back на SettingsActivity, расположенная вверху слева (стрелка назад, рис. 53). Для того, чтобы она работала, нужно добавить строку

```
android:parentActivityName=".MainActivity"
```

в файл манифеста в тэг activity, отвечающий за SettingsActivity, чтобы приложение знало, какая активити в вашем приложении является главной/родительской.

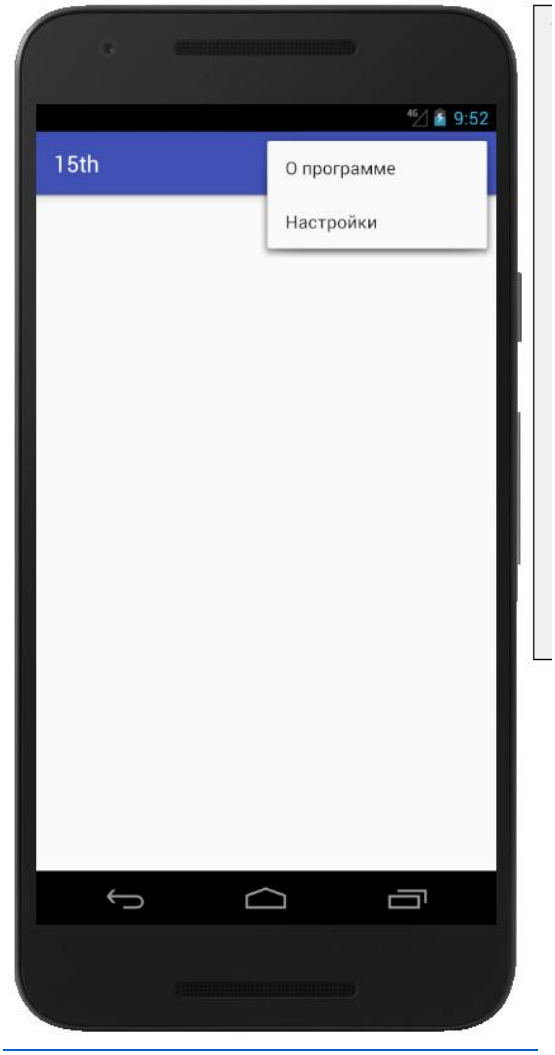

Рисунок 52 – Интерфейс приложения

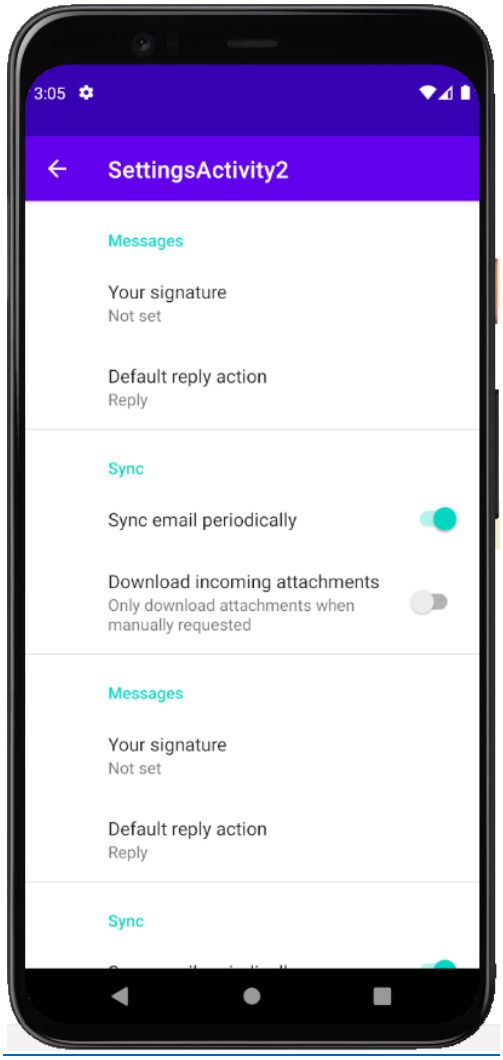

Рисунок 53 – Настройки программы

Однако в этом проекте всё сделано за нас, ни о чём думать не надо, всё уже готово с помощью класса SettingsActivity. Поэтому, чтобы лучше прочувстсвовать механизм SharedPreferences, сделаем следующее. Создадим элемент EditText в интерфейсе приложения. Затем создадим переменную в классе MainActivity:

Java:

```
private EditText edit1;
```

Kotlin:

```
private lateinit var edit1: EditText
```

В методе onCreate класса MainActivity добавим ссылку на переменную и создадим настройки с помощью класса SharedPreferences, после этого тексту в EditText присвоим значение, сохранённое ранее:

Java:

```
edit1 = findViewById(R.id.editText);
SharedPreferences save = getSharedPreferences("SAVE",0);
edit1.setText(save.getString("text",""));
```

Kotlin:

```
edit1 = findViewById(R.id.editTextText)
val save = getSharedPreferences("SAVE", 0)
edit1.setText(save.getString("text", ""))
```

Текст мы будем сохранять по закрытию приложения, поэтому создадим с помощью конструктора метод onStop и добавим туда следующее:

Java:

```
SharedPreferences save = getSharedPreferences("SAVE",0);
SharedPreferences.Editor editor = save.edit(); //создаём редактор shared preferences
editor.putString("text",edit1.getText().toString()); //сохраняем текст из edit1
editor.commit(); //применение редактирования shared preferences
```

Kotlin:

```
val save = getSharedPreferences("SAVE", 0)
val editor = save.edit() //создаём редактор shared preferences
editor.putString("text", edit1.text.toString()) //сохраняем текст из edit1
editor.commit() //применение редактирования shared preferences
```

Проверим работу приложения – запускаем, набираем текст, выходим из приложения, затем запускаем снова и видим, что текст сохранился.

Теперь рассмотрим работу приложения с базой данных. Для этого понадобятся базовые знания SQL.

Перейдите в файл MainActivity.java / kt и в методе onCreate создайте базу данных с названием «DBName»:

Java:

```
SQLiteDatabase db = openOrCreateDatabase("DBname", MODE_PRIVATE, null);
```

Kotlin:

```
val db = openOrCreateDatabase("DBname", MODE_PRIVATE, null)
```

Далее в базе данных нужно создать таблицу:

Java:

```
db.execSQL("CREATE TABLE IF NOT EXISTS MyTable5 (_id INTEGER PRIMARY KEY AUTOINCREMENT, Name VARCHAR);");
```

Kotlin:

```
db.execSQL("CREATE TABLE IF NOT EXISTS MyTable5 (_id INTEGER PRIMARY KEY
AUTOINCREMENT, Name VARCHAR);")
```

И теперь необходимо заполнить созданную таблицу:

Java:

```
db.execSQL("INSERT INTO MyTable5 VALUES ('1','some text');");
```

Kotlin:

```
db.execSQL("INSERT INTO MyTable5 VALUES ('1','some text');")
```

После работы с базой данных её обязательно необходимо закрыть:

Java:

```
db.close();
```

Kotlin:

```
db.close()
```

Запустите приложение, и если не возникло никаких ошибок, значит, создалась база данных, затем в ней создалась таблица и в таблице создалась запись.

Так как не было создано никакого взаимодействия с интерфейсом, то для проверки содержимого базы данных необходимо закомментировать последнюю строку кода, отвечающую за заполнение базы данных (создание первой записи). Далее перед кодом закрытия базы данных необходимо создать указатель:

Java:

```
Cursor cursor = db.rawQuery("SELECT * FROM Mytable5", null);
```

Kotlin:

```
val cursor = db.rawQuery("SELECT * FROM Mytable5", null)
```

И передвинуть его на первую запись:

Java:

```
cursor.moveToFirst();
```

Kotlin:

```
cursor.moveToFirst()
```

Теперь нужно вывести в лог поле Name из первой записи:

Java:

```
Log.d("MEME", cursor.getString(cursor.getColumnIndex("Name")));
```

Kotlin:

```
Log.d("MEME", cursor.getString(cursor.run { getColumnIndex("Name") }))
```

После этого запустите приложение и в Android Studio на нижней панели перейдите в Logcat. Далее в поисковой строке данного раздела (в названии фильтра, где обычно стоит package:mine) нужно написать «MEME» и нажать Enter, после чего по заданному фильтру найдётся нужный нам лог:

2025-02-08 14:42:24.836 5355-5355 MEME site.makse.myapp15ko D some text

Данный лог свидетельствует о том, что была корректно создана база данных, таблица и запись в таблице, а также корректно составлен запрос на извлечение нужных данных (Name='some text', id=1).

Теперь необходимо добавить в приложение работу с базой данных через интерфейсные элементы. Интерфейс нашего приложения почти пустой, этим надо воспользоваться. Например, можно добавить поле EditText и кнопку Button для добавления записи в таблицу, а также ListView и кнопку Button для отображения всех данных в таблице. Можно также добавить кнопки для редактирования и удаления записей из БД – таким образом, мы охватим все наиболее используемые команды языка SQL: SELECT, INSERT, UPDATE, DELETE (или полный CRUD – Create, Read, Update, Delete). Пример интерфейса показан на рисунке 54.

Сформулируем логику приложения. Как показано на рисунке 54, мы хотим, чтобы в приложении можно было добавлять запись в таблицу базы данных с помощью кнопки Add и поля EditText, чтобы можно было просматривать список всех записей в этой таблице, а также - чтобы можно было редактировать запись в таблице и удалять запись. Допустим, что редактировать запись мы будем с помощью диалогового окна, а для отдельной записи в списке ListView предусмотрим элементы TextView и 2 Button для отображения записи и редактирования + удаления записи из таблицы. Таким образом, у нас будет кастомный список ListView и кастомный диалог AlertDialog в приложении. Можно также сделать это приложение с помощью RecyclerView, или LazyColumn, если вы пишете Compose-приложение.

Первое, что нужно сделать, это – нарисовать соответствующий интерфейс для элемента списка и диалога. Но до этого надо добавить в интерфейс приложения (activity_main.xml) кнопку Add справа от уже существующего там поля EditText, и элемент listView снизу от поля и кнопки, который можно найти в Palette в категории Legacy. Создадим в папке Layout 2 файла: list_item.xml и dialog.xml. Файл dialog.xml будет содержать только один элемент – EditText для ввода нового значения записи таблицы базы данных, больше ничего не нужно. Его можно расположить по центру экрана с помощью ConstraintLayout. Пример расположения элементов в list_item.xml с помощью того же ConstraintLayout приведён на рисунке 55.

Рисунок 54 – Пример интерфейса для приложения с БД с listView

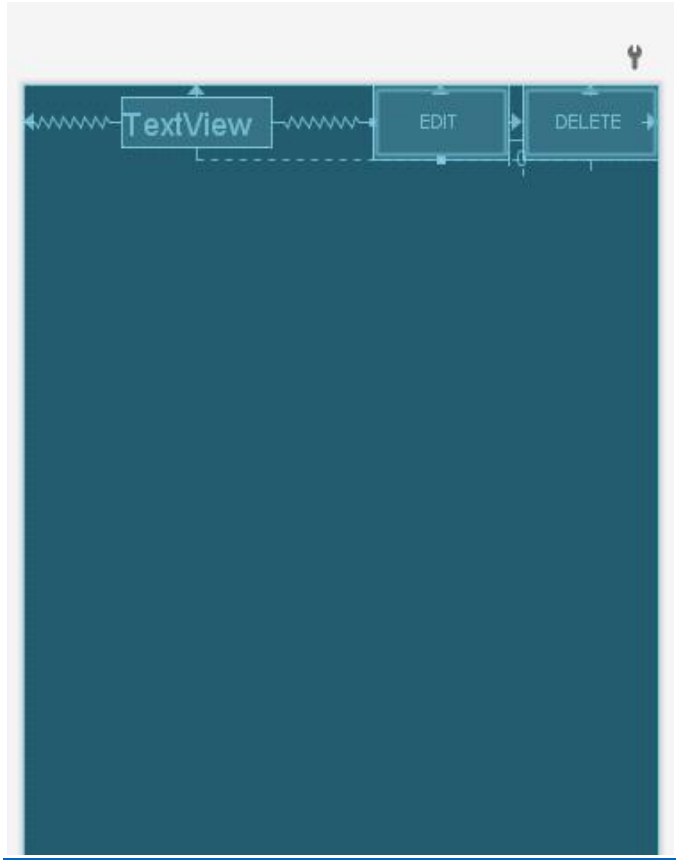

Рисунок 55 – Пример расположения элементов в list_item.xml

Далее в MainActivity создадим некоторые переменные, которые нам понадобятся, и напишем другой код для метода onCreate, закомментировав старый код, относящийся к работе с БД, в котором мы будем загружать данные из БД при загрузке приложения и добавлять новую запись в БД.

Рассмотрим ситуацию с устаревшим элементом listView – для него ниже приведён код на Java. Далее, после окочания этого кода, приведена ситуация с более современным RecyclerView, для которого приведён код на Kotlin.

Java:

```
public class MainActivity extends ListActivity {
   Integer i;
   String[] from;
```

```
   int[] to;
   static ListView listView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      from = new String[]{"Name"};
      to =new int[] {R.id.textView};
      Button btnadd = findViewById(R.id.buttonAdd);
      final EditText editadd = findViewById(R.id.editTextAddingARecord);
      SQLiteDatabase db = openOrCreateDatabase("DBName",MODE_PRIVATE,null);
      db.execSQL("CREATE TABLE IF NOT EXISTS MyTable5 (_id INTEGER PRIMARY KEY AUTOINCREMENT, Name
VARCHAR);");
      Cursor cursor = db.rawQuery("SELECT * FROM Mytable5", null);
      i=cursor.getCount()+1;
      if (cursor.getCount()>0) {
         MyCursorAdapter scAdapter = new
MyCursorAdapter(MainActivity.this,R.layout.list_item,cursor,from,to);
         listView = getListView();
         listView.setAdapter(scAdapter);
      }
      db.close();

      btnadd.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            //TODO

                  }
      });
   }

         }
```

Код метода onClick кнопки ADD будет следующим:

Java:

```
db.execSQL("INSERT INTO MyTable5 VALUES ('"+i+"','"+editadd.getText().toString()+"');");
//i++;
Cursor cursor = db.rawQuery("SELECT * FROM Mytable5", null);
MyCursorAdapter scAdapter = new MyCursorAdapter(MainActivity.this,R.layout.list_item,cursor,from,to);
listView = getListView();
listView.setAdapter(scAdapter);
db.close();
Toast.makeText(getListView().getContext(),"a row added to the table",Toast.LENGTH_LONG).show();
```

Однако, чтобы избежать метаморфоз с PRIMARY KEY при добавлении новой записи после удаления некоторых старых нужно написать дополнительный код в начале этого метода, чтобы найти подходящее значение для id очередной записи, которое удовлетворяло бы ограничению на уникальность. Иначе после удаления 2 записи при наличии 1 и 3 записи в таблице при попытке

добавления очередной записи добавится запись с id=2, после чего при очередном добавлении записи приложение вылетит с ошибкой, т.к. запись с id=3 в таблице уже есть. Этот код выглядит следующим образом:

Java:

```
SQLiteDatabase db = openOrCreateDatabase("DBName",MODE_PRIVATE,null);
Cursor cursor2 = db.rawQuery("SELECT * FROM Mytable5", null);
i=cursor2.getCount()+1;
//цикл для того, чтобы подбирать значения _id и не допускать повторения одинаковых значений (primary key как-никак)
for (int k=1;k<=i;k++) {
   Cursor cursor3 = db.rawQuery("SELECT * FROM Mytable5 WHERE _id="+k+"", null);
   if (cursor3.getCount()==0) {
      i=k;
      break;
   }
}
```

Обратите внимание, что ListView в интерфейсе activity_main.xml необходимо присвоить id «@android:id/list». Сделать это можно в текстовом представлении, заменив в тэге ListView старую строку на следующую:

```
android:id="@android:id/list"
```

И, как Вы уже заметили, для того, чтобы приведённый выше код работал, нужно создать дополнительный класс MyCursorAdapter, занимающийся обработкой нажатий кнопок EDIT и DELETE в интерфейсе приложения для каждой отдельной записи (лучше всего его создать в конце файла, после кода класса MainActivity):

Java:

```
public class MyCursorAdapter extends SimpleCursorAdapter {
   private int layout_;
   private Cursor cursor;
   String[] from;
   int[] to;
   ListView listView;
   EditText edit2;

   public MyCursorAdapter(Context context, int layout, Cursor c, String[] from, int[] to) {
      super(context, layout, c, from, to);
      layout_=layout;
      cursor=c;
   }

   @Override
   public void bindView(View view, Context _context, Cursor cursor) {
      String data = cursor.getString(cursor.getColumnIndex("Name"));
```

```
        int id = cursor.getInt(cursor.getColumnIndex("_id"));
        TextView text = view.findViewById(R.id.textViewListItemText);
        text.setText(data);
        Button butdel = view.findViewById(R.id.buttonDelete);
        Button butedit = view.findViewById(R.id.buttonEdit);
        listView = MainActivity.listView;
        butdel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                SQLiteDatabase db = _context.openOrCreateDatabase("DBName",MODE_PRIVATE,null);
                db.execSQL("DELETE FROM MyTable5 WHERE _id="+id+"");
                Cursor cursor = db.rawQuery("SELECT * FROM Mytable5", null);
                from = new String[]{"Name"};
                to =new int[] {R.id.textView};
                MyCursorAdapter scAdapter = new MyCursorAdapter(_context,R.layout.list_item,cursor,from,to);
                listView.setAdapter(scAdapter);
                db.close();
                Toast.makeText(_context,"row deleted from the db id="+id,Toast.LENGTH_LONG).show();
            }
        });
        butedit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                AlertDialog.Builder dialog = new AlertDialog.Builder(_context);
                dialog.setMessage("Enter new value:");
                dialog.setTitle("Changing the item");
                LayoutInflater inflater = new LayoutInflater(_context) {
                    @Override
                    public LayoutInflater cloneInContext(Context context) {
                        return null;
                    }
                };
                View dialogview = inflater.inflate(R.layout.dialog,null);
                dialog.setView(dialogview);
                edit2 = dialogview.findViewById(R.id.editTextCnahgeDBRecord);
                edit2.setText(text.getText().toString());
                dialog.setNeutralButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int i) {
                        SQLiteDatabase db = _context.openOrCreateDatabase("DBName",MODE_PRIVATE,null);
                        db.execSQL("UPDATE MyTable5 SET Name='"+edit2.getText().toString()+"' WHERE _id="+id+"");
                        Cursor cursor = db.rawQuery("SELECT * FROM Mytable5", null);
                        from = new String[]{"Name"};
                        to =new int[] {R.id.textView};
                        MyCursorAdapter scAdapter = new
MyCursorAdapter(_context,R.layout.list_item,cursor,from,to);
                        listView.setAdapter(scAdapter);
                        db.close();
                        Toast.makeText(_context,"row edited from the db row id="+id,Toast.LENGTH_LONG).show();
                        dialog.dismiss();
                    }
                });
                dialog.setIcon(R.mipmap.ic_launcher_round);
                AlertDialog alertDialog = dialog.create();
```

```
            alertDialog.show();
        }
    });

}

@Override
public View newView(Context context, Cursor cursor, ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater) context.getSystemService(context.LAYOUT_INFLATER_SERVICE);
    View view = inflater.inflate(layout_,parent,false);
    return view;
}
}
```

Возможно, возникнет проблема с методом cursor.getColumnIndex - его можно заменить по совету Android Studio на метод cursor.getColumnIndexOrThrow.

Также можно всё – и Shared Preferences, и базу данных сделать в одном приложении, поменяв extends ListActivity в классе MainActivity.java обратно на AppCompatActivity и подумав, как исправить ошибки.

Чтобы при вводе с клавиатуры она не перемещала кнопки и всё остальное вверх при своём появлении внизу экрана, можно указать параметр

```
android:windowSoftInputMode="adjustPan"
```

в файле манифеста в тэге главной активити нашего приложения.

Теперь рассмотрим вариант с RecyclerView и Kotlin. Надеюсь, вы уже умеете добавлять и настраивать нужным образом элемент RecyclerView в интерфейс приложения, см. рисунок 56. Если нет, можно посмотреть в лабораторную 5 выше, чтобы вспомнить, как это делается.

Первое, что нам понадобится после добавления RecyclerView в интерфейс и задания для него нужных свойств, это – дополнительный класс MyCursorAdapter, содержащий функционал для работы адаптера RecyclerView. Создадим новый класс в папке с названием нашей package, и сделаем ему следующий заголовок:

Kotlin:

```
class MyCursorAdapter(
    context: Context?, private var records: List<String>
) :
    RecyclerView.Adapter<MyCursorAdapter.MyViewHolder>() {

    var context = context
```

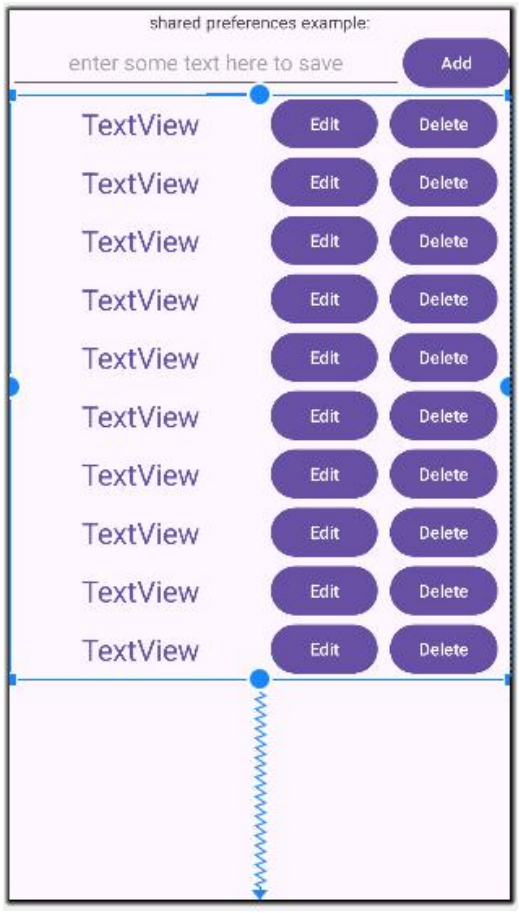

Рисунок 56 – Пример интерфейса для приложения с БД с RecyclerView

Видно, что мы будем передавать этому кастомному адаптеру уже готовый список строк из БД. Переменная context нам понадобится в дальнейшем.

Далее, при таком заголовке Android Studio сама должна предложить создать 3 метода, код которых приведён ниже:

Kotlin:

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

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

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

    holder.text.text = records[position]

    val db = context!!.openOrCreateDatabase("DBName", MODE_PRIVATE, null)
    var cursor = db.rawQuery("SELECT * FROM Mytable5", null)
    cursor.moveToFirst()
    cursor.move(position)
    val id = cursor.getInt(cursor.run { getColumnIndex("_id") })
```

```
    cursor.close()
    db.close()

    holder.butdel.setOnClickListener {
        val db = context!!.openOrCreateDatabase("DBName", MODE_PRIVATE, null)
        db.execSQL("DELETE FROM MyTable5 WHERE _id=$id")
        cursor.close()
        db.close()
        Toast.makeText(context, "row deleted from the db id=$id",
Toast.LENGTH_LONG).show()


        this.notifyItemRemoved(position)
        this.notifyItemRangeChanged(0,records.size)
        Update()
    }

    holder.butedit.setOnClickListener {
        val dialog = AlertDialog.Builder(context)
        dialog.setMessage("Enter new value:")
        dialog.setTitle("Changing the item")

        val dialogview: View =
LayoutInflater.from(context).inflate(R.layout.dialog, null)
        dialog.setView(dialogview)
        val edit2 = dialogview.findViewById<EditText>(R.id.editTextChangeDBRecord)
        edit2.run { this!!.setText(holder.text!!.text.toString()) }
        dialog.setNeutralButton(
            "OK"
        ) { dialog, i ->
            val db = context!!.openOrCreateDatabase(
                "DBName",
                MODE_PRIVATE,
                null
            )
            db.run {
                execSQL(
                    "UPDATE MyTable5 SET Name='" + edit2.run { this!!.getText() }
+ "' WHERE _id=" + id + ""
                )
            }
            cursor.close()
            db.close()

            this.notifyItemChanged(position)
            Update()

            Toast.makeText(context, "row edited from the db row id=$id",
Toast.LENGTH_LONG)
                .show()
            dialog.dismiss()
        }
        dialog.setIcon(R.mipmap.ic_launcher_round)
        val alertDialog = dialog.create()
        alertDialog.show()

    }


}
```

Метод `onBindViewHolder` такой длинный, потому что в нём находятся обработчики событий нажатия кнопок Edit и Delete. Обратите внимание, что после удаления или обновления записи в БД и,

соответственно, в списке, мы должеы уведомить RecyclerView, являющийся observer для нашего адаптера, что произошли соответствующие изменения – это делается с помощью методов `this.notifyItemChanged(position)`, `this.notifyItemRemoved(position)` и `this.notifyItemRangeChanged(0,records.size)`.

Далее – Вы наверняка заметили, что не хватает ещё кое-чего. Во-первых, нам нужен собственный кастомный класс ViewHolder для нашего списка, который содержит ссылки на нужные интерйфейсные элементы нашего элемента списка:

Kotlin:

```
class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val text: TextView = itemView.findViewById<TextView>(R.id.textViewListItem)
    val butdel = itemView.findViewById<Button>(R.id.deleteButton)
    val butedit = itemView.findViewById<Button>(R.id.editButton)
}
```

И ещё нам нужен метод Update(), который обновляет сам список строк `records` и наш observer RecyclerView в соответствии с изменённым содержимым БД. Это не лучший, но работающий способ добиться нужного функционала:

Kotlin:

```
fun Update() {
    val db = context!!.openOrCreateDatabase("DBName", MODE_PRIVATE, null)
    var cursor = db.rawQuery("SELECT * FROM Mytable5", null)
    Log.i("numberofrecordsindb", cursor.count.toString())
    val data = mutableListOf<String>()
    var i:Int = 0
    while (cursor.moveToNext()) {
        data.add(cursor.getString(cursor.getColumnIndexOrThrow("Name")))
        i++
    }
    cursor.close()
    db.close()
    records=data
}
```

И теперь всё, что нам нужно, это написать нужный код в классе MainActivity.kt:

Kotlin:

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        edit1 = findViewById(R.id.editTextText)
        val save = getSharedPreferences("SAVE", 0)
        edit1.setText(save.getString("text", ""))

//        val db = openOrCreateDatabase("DBname", MODE_PRIVATE, null)
//        db.execSQL("CREATE TABLE IF NOT EXISTS MyTable5 (_id INTEGER PRIMARY KEY
AUTOINCREMENT, Name VARCHAR);")
//        db.execSQL("INSERT INTO MyTable5 VALUES ('1','some text');")
//        val cursor = db.rawQuery("SELECT * FROM Mytable5", null)
//        cursor.moveToFirst()
//        Log.d("MEME", cursor.getString(cursor.run { getColumnIndex("Name") }))
//        db.close()
```

```
        val btnadd = findViewById<Button>(R.id.addButton)


        val editadd = edit1
        val db = openOrCreateDatabase("DBName", MODE_PRIVATE, null)
        db.execSQL("CREATE TABLE IF NOT EXISTS MyTable5 (_id INTEGER PRIMARY KEY
AUTOINCREMENT, Name VARCHAR);")
        val cursor = db.rawQuery("SELECT * FROM Mytable5", null)
        var i = cursor.count + 1
        if (cursor.count > 0) {
            val scAdapter = MyCursorAdapter(this@MainActivity, fillList())
            val recyclerView = findViewById<RecyclerView>(R.id.review1)
            recyclerView.layoutManager = LinearLayoutManager(this)
            recyclerView.adapter = scAdapter
        }
        cursor.close()
        db.close()

        btnadd.setOnClickListener {
            val db = openOrCreateDatabase("DBName", MODE_PRIVATE, null)
            val cursor2 = db.rawQuery("SELECT * FROM Mytable5", null)
            i = cursor2.count + 1
            cursor2.close()

//цикл для того, чтобы подбирать значения _id и не допускать повторения одинаковых
значений (primary key как-никак)
            for (k in 1..i!!) {
                val cursor3 = db.rawQuery("SELECT * FROM Mytable5 WHERE _id=$k",
null)
                if (cursor3.count == 0) {
                    i = k
                    break
                }
                cursor3.close()
            }

            db.execSQL(("INSERT INTO MyTable5 VALUES ('" + i + "','" +
editadd.text.toString()).toString() + "');")
            //i++;
            val cursor = db.rawQuery("SELECT * FROM Mytable5", null)
            val scAdapter = MyCursorAdapter(this@MainActivity, fillList())
            val recyclerView = findViewById<RecyclerView>(R.id.review1)
            recyclerView.adapter = scAdapter
            cursor.close()
            db.close()
            Toast.makeText(this@MainActivity, "a row added to the table",
Toast.LENGTH_LONG)
                .show()

        }


    }
```

В коде выше есть обработчик нажатия кнопки Add для добавления записи в БД. И последнее, что нам нужно, это написать аналог функции Update() из класса MyCursorAdapter, только теперь в классе MainActivity.kt:

Kotlin:

```
private fun fillList(): List<String> {
    val db = this.openOrCreateDatabase("DBName", MODE_PRIVATE, null)
    var cursor = db.rawQuery("SELECT * FROM Mytable5", null)
    Log.i("numberofrecordsindb", cursor.count.toString())
    val data = mutableListOf<String>()
    var i:Int = 0
    while (cursor.moveToNext()) {
        data.add(cursor.getString(cursor.getColumnIndexOrThrow("Name")))
        i++
    }
    cursor.close()
    db.close()
    return data
}
```

В результате выполнения этого кода получим приложение, изображённое на рисунке 57 – всё работает. В качестве практики можно подумать, как обойтись без метода Update в нашем кастомном классе адаптера.

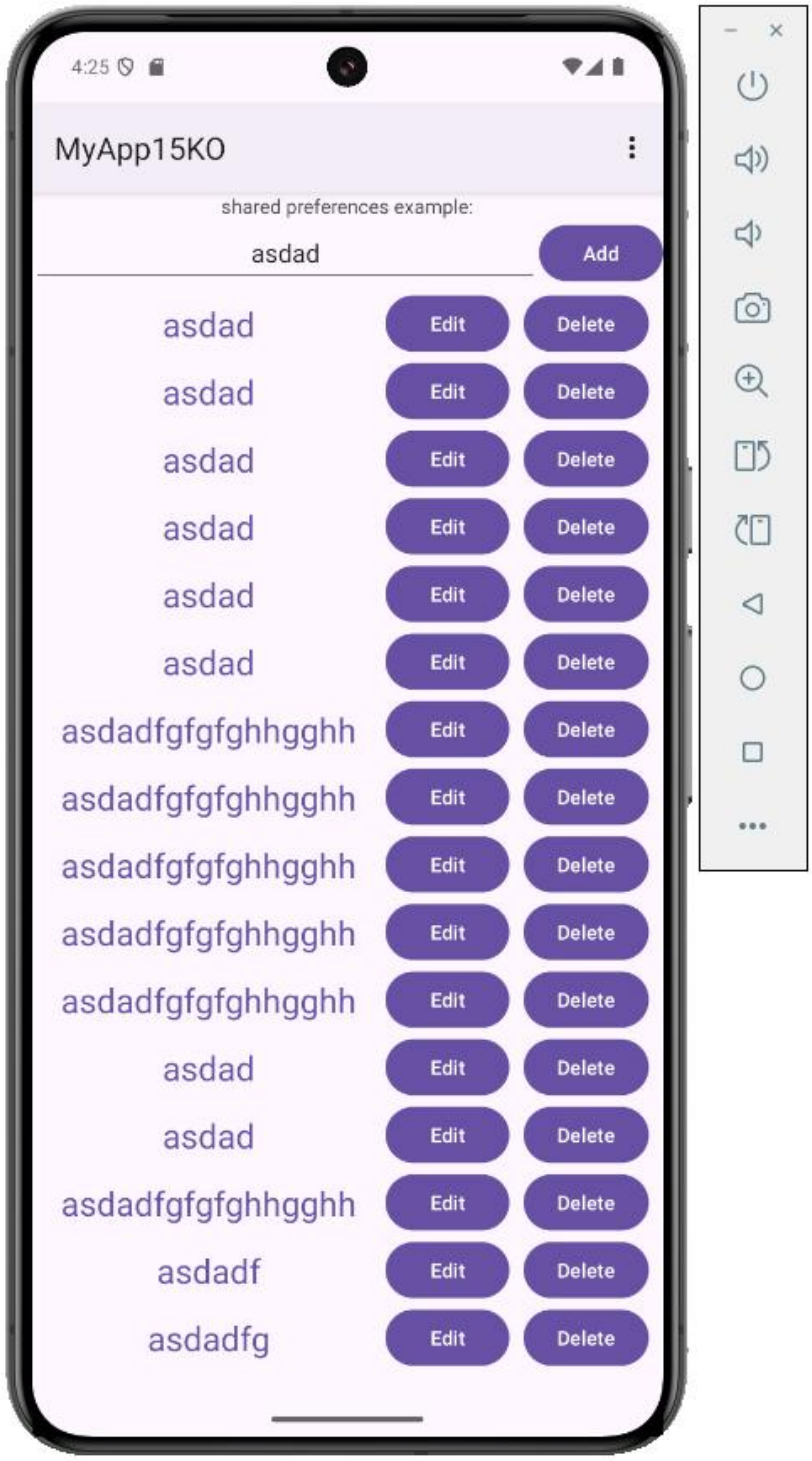

Рисунок 57 – Пример работающего приложения с БД с RecyclerView

И всё - приложение, работающее с базой данных и включающее основные операции SQL завершено. Единственное, что нужно помнить – это приложение не идеально; все операции с записями хорошо бы перенести в отдельный класс (как и SELECT, и INSERT), причём – в отдельные методы этого класса, называющиеся соответствующим образом; затем – интерфейс, конечно же, нужно загружать в отдельном асинхронном потоке, т.к. – представьте, что у Вас в таблице 1000 строк, а не 10-15; что тогда будет с работой приложения?

Лабораторная работа №16. Облачное мобильное приложение

Задание: Создать простейшее облачное мобильное приложение, используя Firebase и Android Studio.

Для выполнения этого задания можно выбрать любой облачный сервис, будь то push-уведомления, база данных реального времени, аналитика мобильного приложения (не в Google Play, а с помощью сторонних средств, например, Flurry, MixPanel, Crashlytics и т.д.), облачное хранилище, использование облачных сервисов, связанных с тестированием, распознаванием изображений и т.д. Рассмотрим один из простых примеров – создание push-уведомлений с помощью Firebase.

Первое, что нужно сделать – это создать новый проект в консоли Firebase (https://console.firebase.google.com/). Добавить проект – назовите как-то проект, выберите местоположение и нажмите на чекбокс "Я принимаю", а затем – на кнопку Создать проект, см. рисунок 46. Затем нажмите на Продолжить.

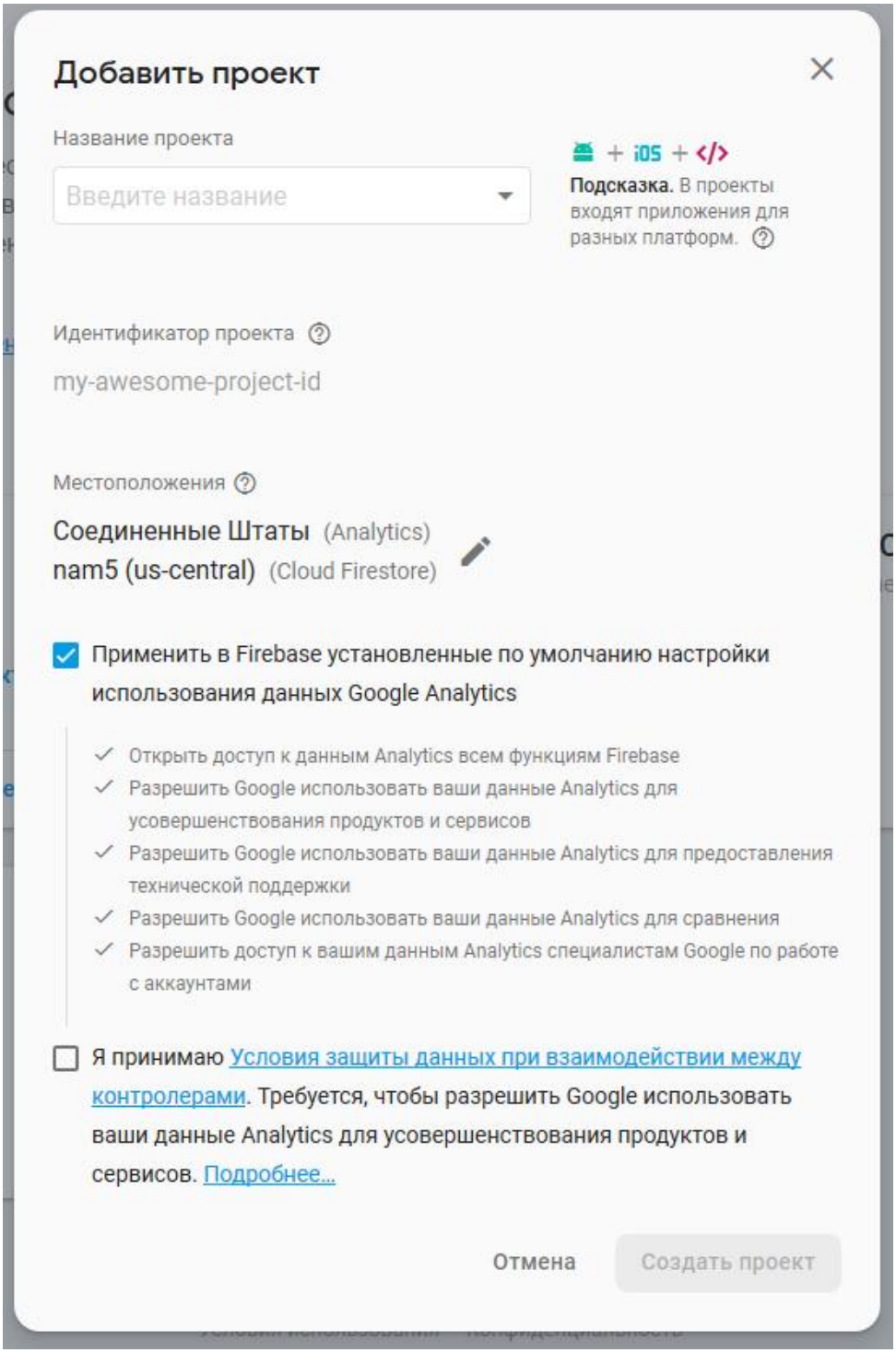

Рисунок 46 – Создание проекта в Firebase

Откроется окно управления проектом, в котором нам нужно нажать на кнопку с иконкой ОС Андроид над надписью Добавьте приложение. После этого нам потребуется ввести название пакета, поэтому перейдём в Android Studio и создадим новый проект с Empty Activity. Добавим в проект опциональное меню с пунктом вызова окна О программе, как обычно. Затем перейдём в файл манифеста и скопируем значение переменной package, которое вставим в окно Firebase в строку

Название пакета Android. Дополнительно можно заполнить поля псевдонима и хэша сертификата, но это необязательно. Нажмём на кнопку Зарегистрировать приложение.

Далее скачаем файл конфигурации google-services.json и поместим его в корень нашего проекта. Для этого переключим вид в Project Explorer (слева вверху) с Android на Project, щёлкнем правой кнопкой мыши по папке app и нажмём Paste, и в открывшемся окне – OK. В режиме Project мы увидим файл json, а в режиме Android его не видно. Затем переключитесь обратно в режим Android.

В окне Firebase браузера нажмём Далее.

Теперь надо изменить файлы gradle. Следуйте инструкциям в окне браузера, так как там может быть более новая версия, чем здесь. Итак, в файл build.gradle уровня проекта (Project: название) в dependencies надо добавить строку

```
classpath 'com.google.gms:google-services:4.0.1'
```

В файл build.gradle уровня приложения (Module: app) добавьте следующее в секцию dependencies:

```
implementation 'com.google.firebase:firebase-core:16.0.1'
implementation 'com.google.firebase:firebase-messaging:17.4.0'
```

и в самый конце этого файла:

```
apply plugin: 'com.google.gms.google-services'
```

После этого синхронизируйте проект (Sync Now). В окне браузера нажмите кнопку Далее. После этого обязательно запустите приложение! При этом эмулятор или реальное устройство должны быть подлючены к интернету. Если всё в порядке, в окне проверки связи с приложением браузера будет выведена соответствующая зелёная надпись (Поздравляем! Вы добавили сервис Firebase в свое приложение). Если связи с приложением нет, повторите предыдущие шаги.

Теперь удалим TextView "Hello World!", добавим в интерфейс на его место кнопку, и напишем внутри обработчика нажатия на неё следующий код:

```
String tkn = FirebaseInstanceId.getInstance().getToken();
Toast.makeText(MainActivity.this, "Current token ["+tkn+"]",
    Toast.LENGTH_LONG).show();
Log.d("App", "Token ["+tkn+"]");
```

Добавим в папку с MainActivity.java следующий класс в отдельном java-файле FireIDService.java:

```
public class FireIDService extends FirebaseInstanceIdService {
  @Override
  public void onTokenRefresh() {
    String tkn = FirebaseInstanceId.getInstance().getToken();
    Log.d("Not","Token ["+tkn+"]");
  }
}
```

А затем добавим следующий класс, отвечающий за сервис, обрабатывающий входящее push-уведомление:

```
public class FireMsgService extends FirebaseMessagingService {

  @Override
  public void onMessageReceived(RemoteMessage remoteMessage) {
    super.onMessageReceived(remoteMessage);

    Log.d("Msg", "Message received ["+remoteMessage+"]");

    // Create Notification
    Intent intent = new Intent(this, MainActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 1410,
        intent, PendingIntent.FLAG_ONE_SHOT);

    NotificationCompat.Builder notificationBuilder = new
        NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.common_full_open_on_phone)
        .setContentTitle("Message")
        .setContentText(remoteMessage.getNotification().getBody())
        .setAutoCancel(true)
        .setContentIntent(pendingIntent);

    NotificationManager notificationManager =
        (NotificationManager)
            getSystemService(Context.NOTIFICATION_SERVICE);

    notificationManager.notify(1410, notificationBuilder.build());
  }
}
```

И последнее, что нужно сделать – добавить в файл манифеста соответствующий сервис внутри тэга application:

```
        <service android:name=".FireIDService">
    <intent-filter>
      <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>
```

Теперь постройте apk-файл и установите приложение на эмуляторе или на реальном устройстве. После этого нажмите в браузере на кнопку Открыть консоль, нажмите на название своего проекта и выберите кнопку настроек справа от названия. Откроется окно настроек проекта, в котором на закладке Cloud Messaging можно увидеть ключ сервера для аутентификации. Зная этот ключ, можно посылать сообщения внутри приложения, с помощью другого приложения и с помощью самой консоли Firebase, при этом планируя их на определённое время, например, и для определённой аудитории.

Чтобы отправить сообщение, перейдите в категорию Grow -> Cloud Messaging и нажмите на кнопку Send your first message. Заполните поля (текст сообщения), нажмите кнопку Далее. Выберите приложение на следующем шаге (Аудитория) – при этом Вы можете выбирать сегмент пользователей – либо все пользователи, у которых установлено приложение, либо конкретного пользователя, либо – пользователей, подписанных на определённую тему. Нажмите кнопку Далее. В секции Планирование можно настроить расписание, когда пользователи получат сообщение, либо просто оставить Now для немедленной отправки сообщения. Нажмите Далее. В следующей секции можно задавать события-конверсии, чтобы понимать, откуда приходят пользователи – это больше связано с аналитикой приложения. Если перейти с помощью кнопки Далее к следующей секции, можно настрогить дополнительные параметры, например – ключ конкретного пользователя, который получит уведомление. Для завершения нажмите кнопку Проверить. Вы увидите подтверждающее окно с уведомлением, в котором надо нажать кнопку опубликовать (рис. 47).

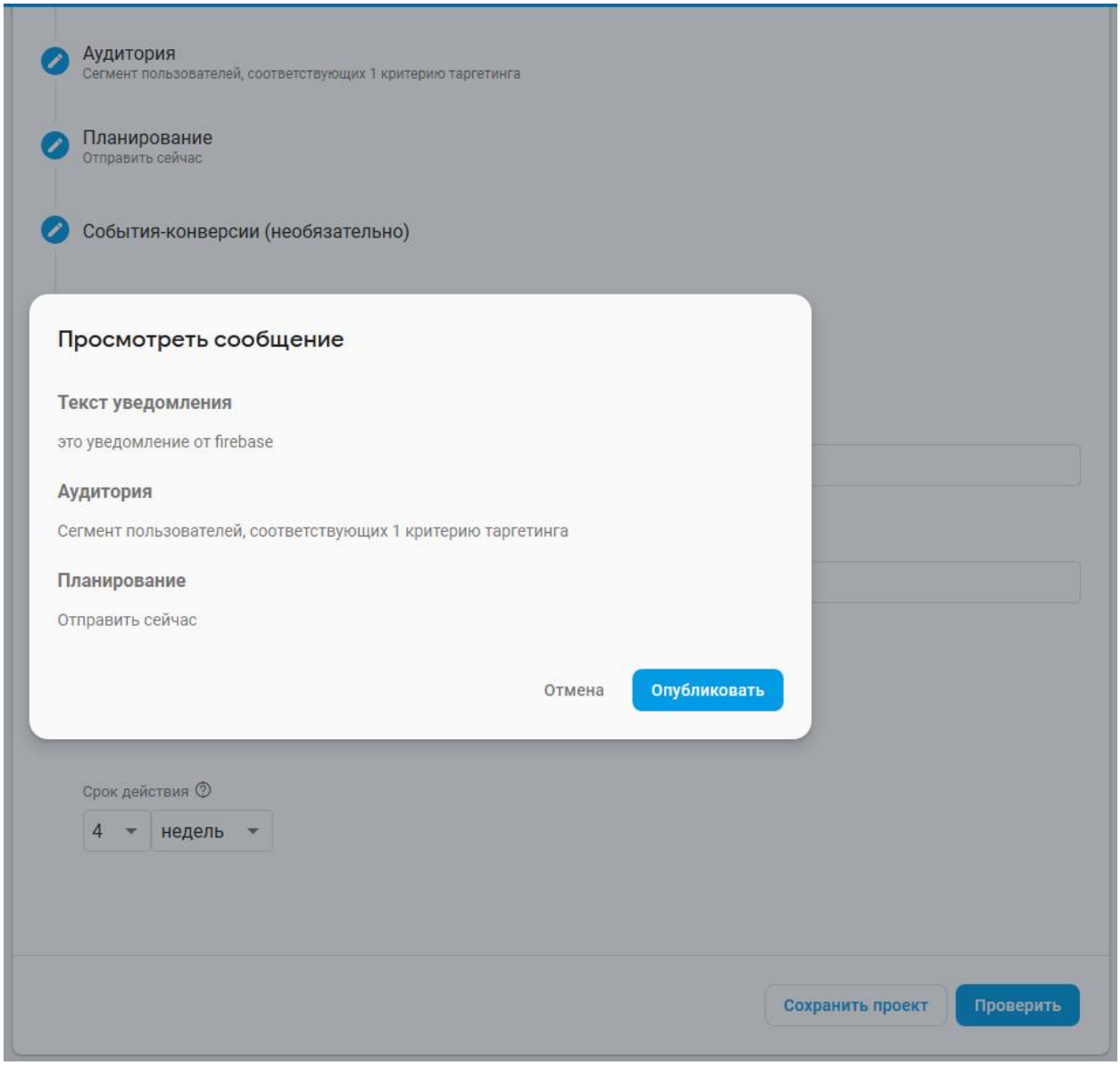

Рисунок 47 – Публикация push-уведомления с помощью Firebase

После нажатия на кнопку Опубликовать, даже если у Вас приложение не запущено на мобильном устройстве, появится уведомление с небольшой картинкой и стандартным звуком уведомления для Вашего устройства. В панели уведомлений Вы увидите тему уведомления и текст. При нажатии на уведомление запустится Ваше приложение. При выключенном интернете Вы увидите уведомление сразу, как только подключитесь к интернету. При этом консоль Firebase переключится на страницу с уведомлениями, где отобразится созданное уведомление с указанием некоторых подробностей, и где можно будет его продублировать, создать новое и создать эксперимент с определённой целевой аудиторией (рис. 48).

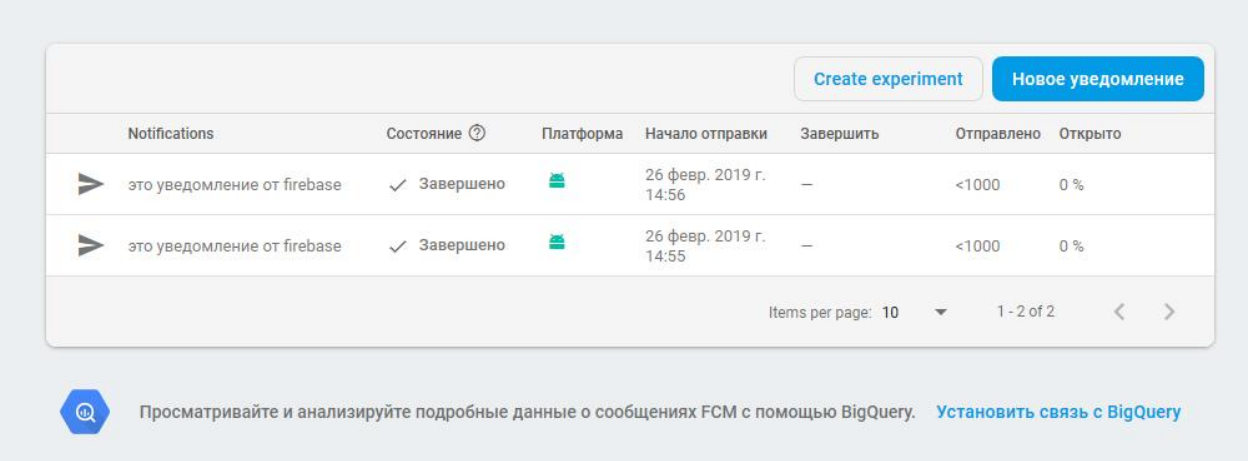

Рисунок 48 – Результат публикации push-уведомления

Можно написать приложение, в котором будут 2 кнопки: одна будет подписываться на уведомления, а другая – формировать и присылать уведомление. Можно написать 2 приложения – одно отсылает уведомления, другое принимает. А можно использовать какой-то другой механизм работы с облаком, например, выводить в приложение аналитику по его крашам у пользователей с помощью Firebase.