

Хранение данных. ContentProvider

Хайминов Алексей



Хайминов Алексей



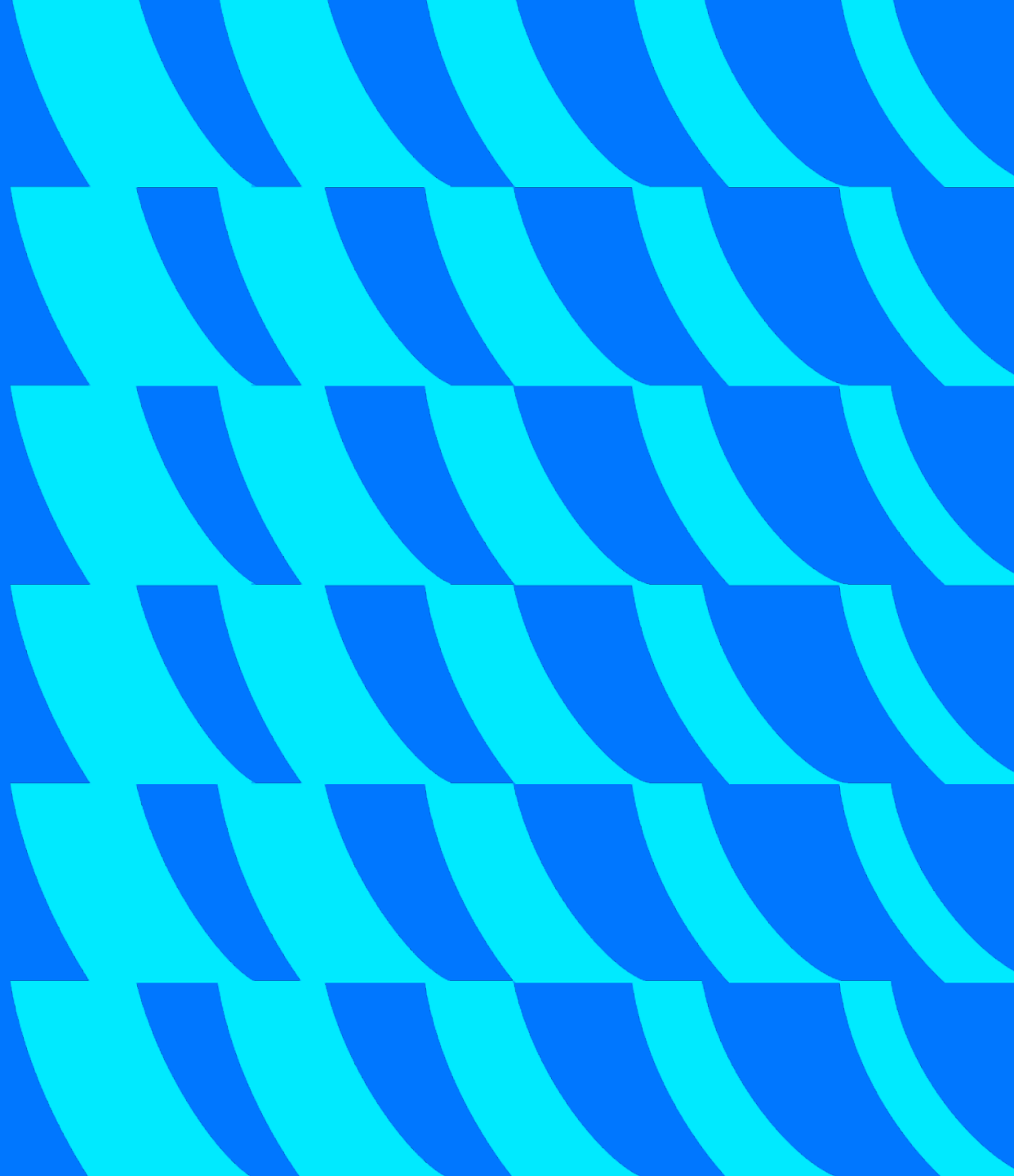
- С 2013 года - Android
- «Банк Софт Системс»
- «Одноклассники»
- стартапы

- Сейчас - руководитель функции Android, проекты:
- «Юла»
- «ВКонтакте» - команды бизнес-юнита СМБ

Занятие

- Зачем?
- Shared Preferences
- DataStore
- Files
- AccountManager
- SQLite
- ContentProvider
- Room

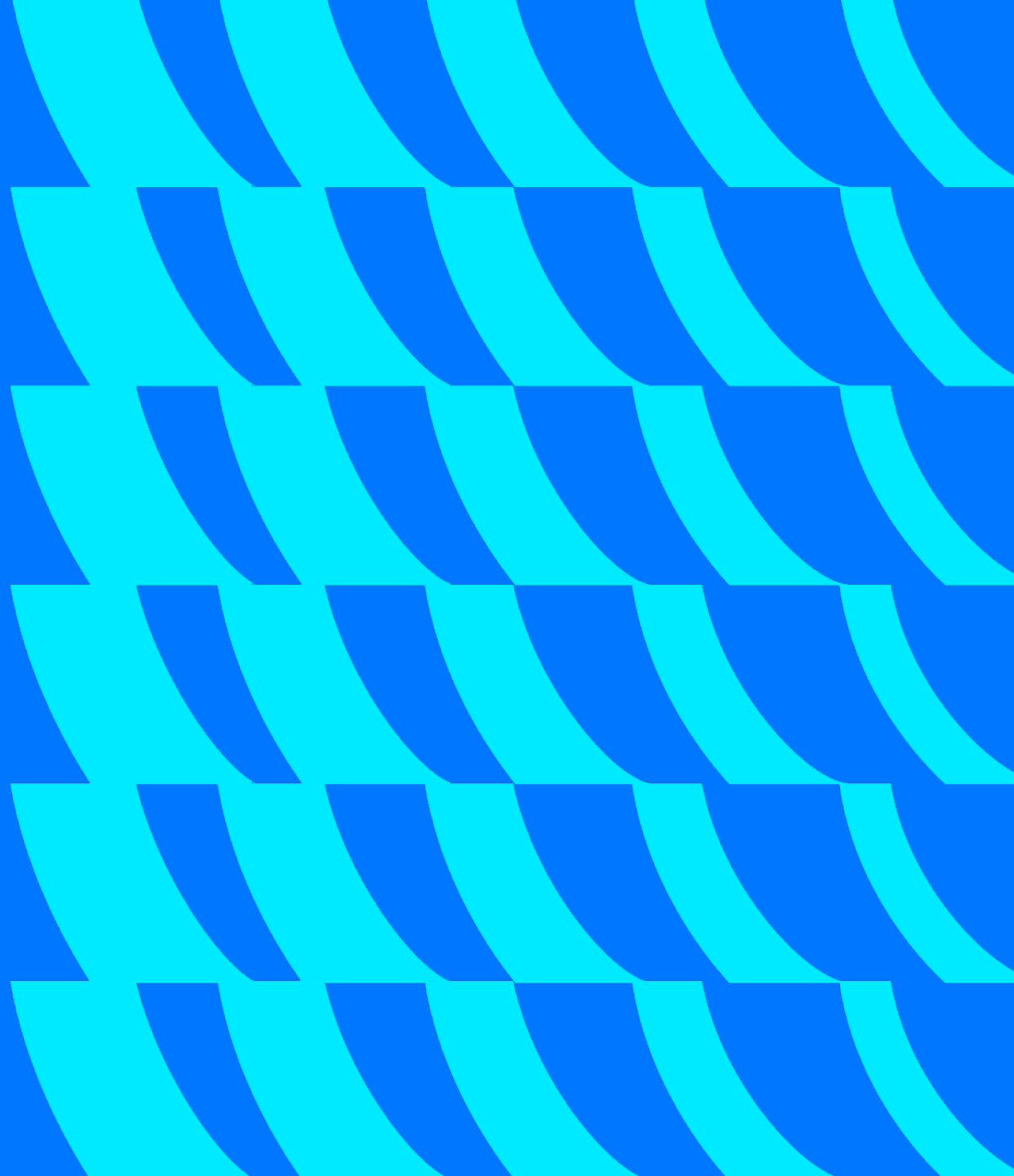
Организационные моменты





Напоминание
отметиться на
портале

Зачем?



Какие данные храним

1

Кеширование
контента вашего API

2

Настройки
приложения
(например тема,
шрифт)

3

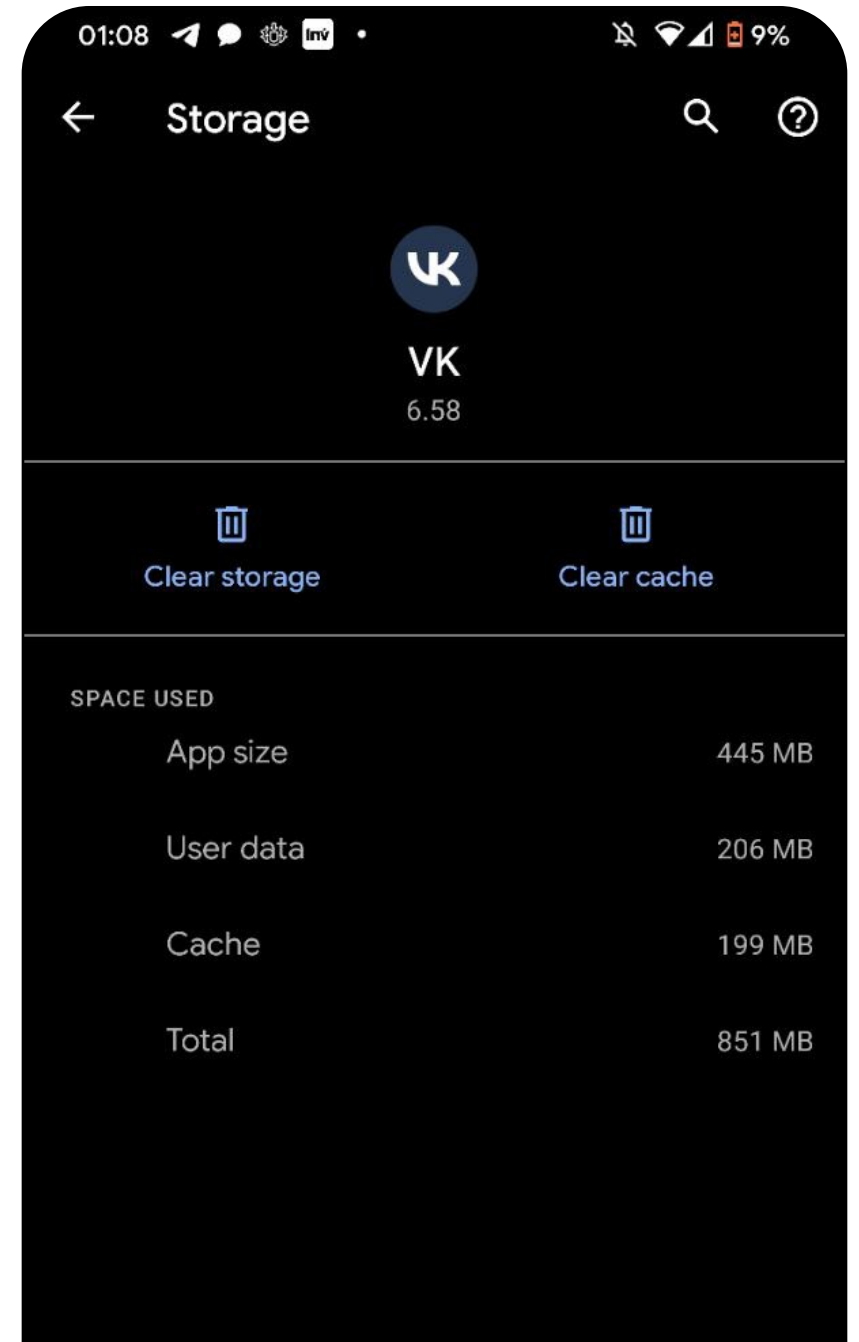
Авторизация/токены/
device_id/etc

4

Контент, который
создает сам юзер
(фоторедактор etc)

Проблемы с хранением данных

- Пользователь может очистить данные и кеш в приложении
- Бесконечный рост данных
- Безопасность хранения и ограничение доступа
- Доступность данных

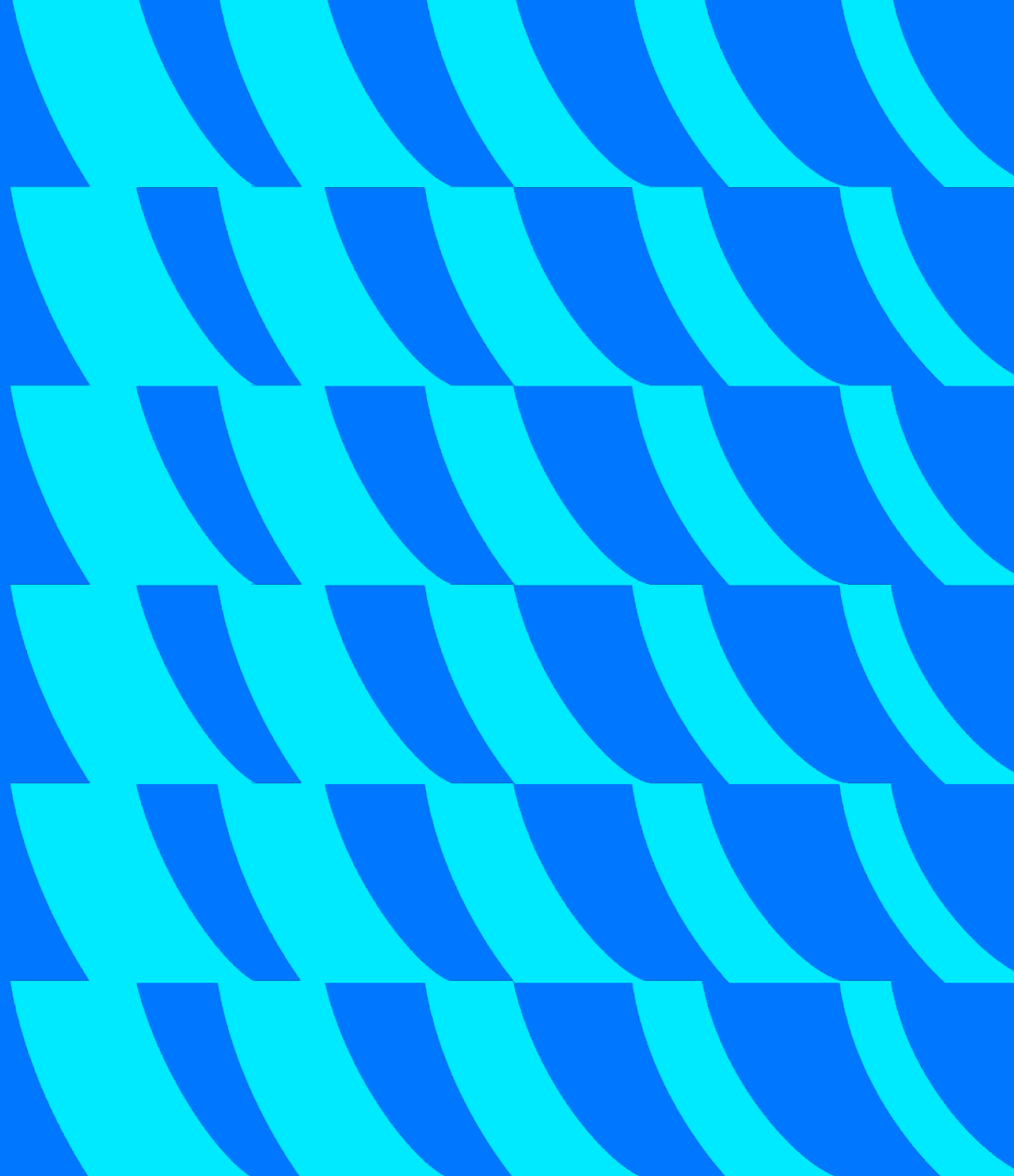


Время доступа к данным

0.16 0.16 0.17 0.15 0.15 **2.5** 0.17 0.16 0.16 0.16 0.15 0.17 0.17 0.17 0.19 0.19 0.17
0.2 0 0 **4.79** 0.24 0.23 0.28 0.23 0.28 0.03 0 11.23 0.16 0.15 0.18 0.16 0.16 0.17
0.16 **5.84**

Пишем в файл строку

Shared Preferences



Shared Preferences

В `SharedPreferences` можно хранить следующие типы данных:

- Примитивные типы (`boolean`, `float`, `int`, `long`)
- Строки (`String`)
- Множества строк (`Set<String>`)

Shared Preferences

Пример работы чтение -

```
const val PREFS_NAME = "MyPrefsFile"
```

```
val settings = getSharedPreferences(PREFS_NAME, 0)
```

```
val silent = settings.getBoolean("silentMode", false)
```

Запись -

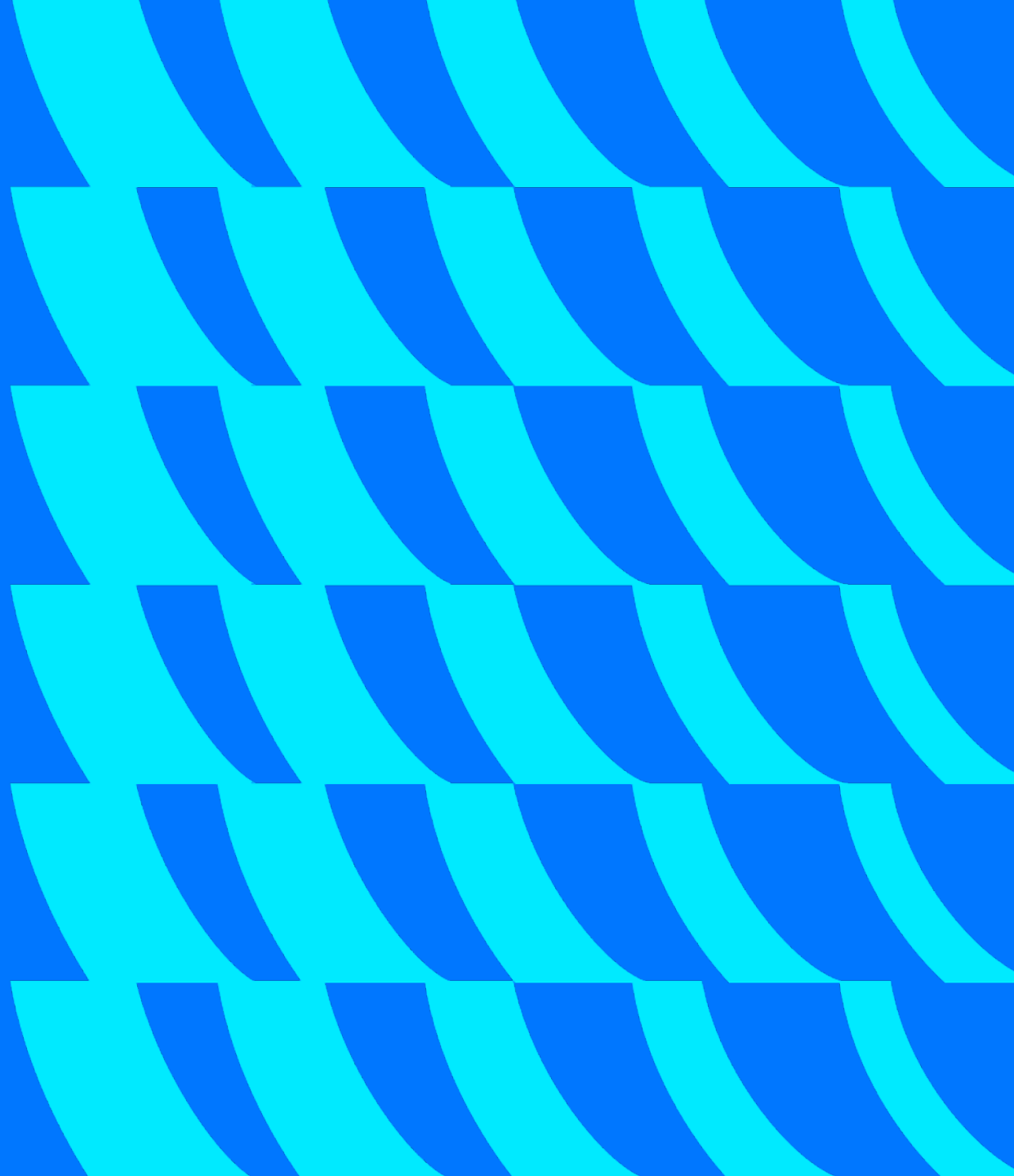
```
val settings = getSharedPreferences(PREFS_NAME, 0)
```

```
val editor = settings.edit()
```

```
editor.putBoolean("silentMode", silentMode)
```

```
editor.apply()
```

Data Store



DataStore - Preferences

// At the top level of your kotlin file:

```
val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = «settings»)
```

//Чтение

```
val EXAMPLE_COUNTER = intPreferencesKey("example_counter")
val exampleCounterFlow: Flow<Int> = context.dataStore.data
    .map { preferences ->
        // No type safety.
        preferences[EXAMPLE_COUNTER] ?: 0
    }
}
```

//Запись

```
suspend fun incrementCounter() {
    context.dataStore.edit { settings ->
        val currentCounterValue = settings[EXAMPLE_COUNTER] ?: 0
        settings[EXAMPLE_COUNTER] = currentCounterValue + 1
    }
}
```

DataStore - Proto

1. Protobuf File (*.proto) - /app/src/main/proto

```
syntax = "proto3";

option java_package = "com.example.application";
option java_multiple_files = true;

message Settings {
    int32 example_counter = 1;
}
```

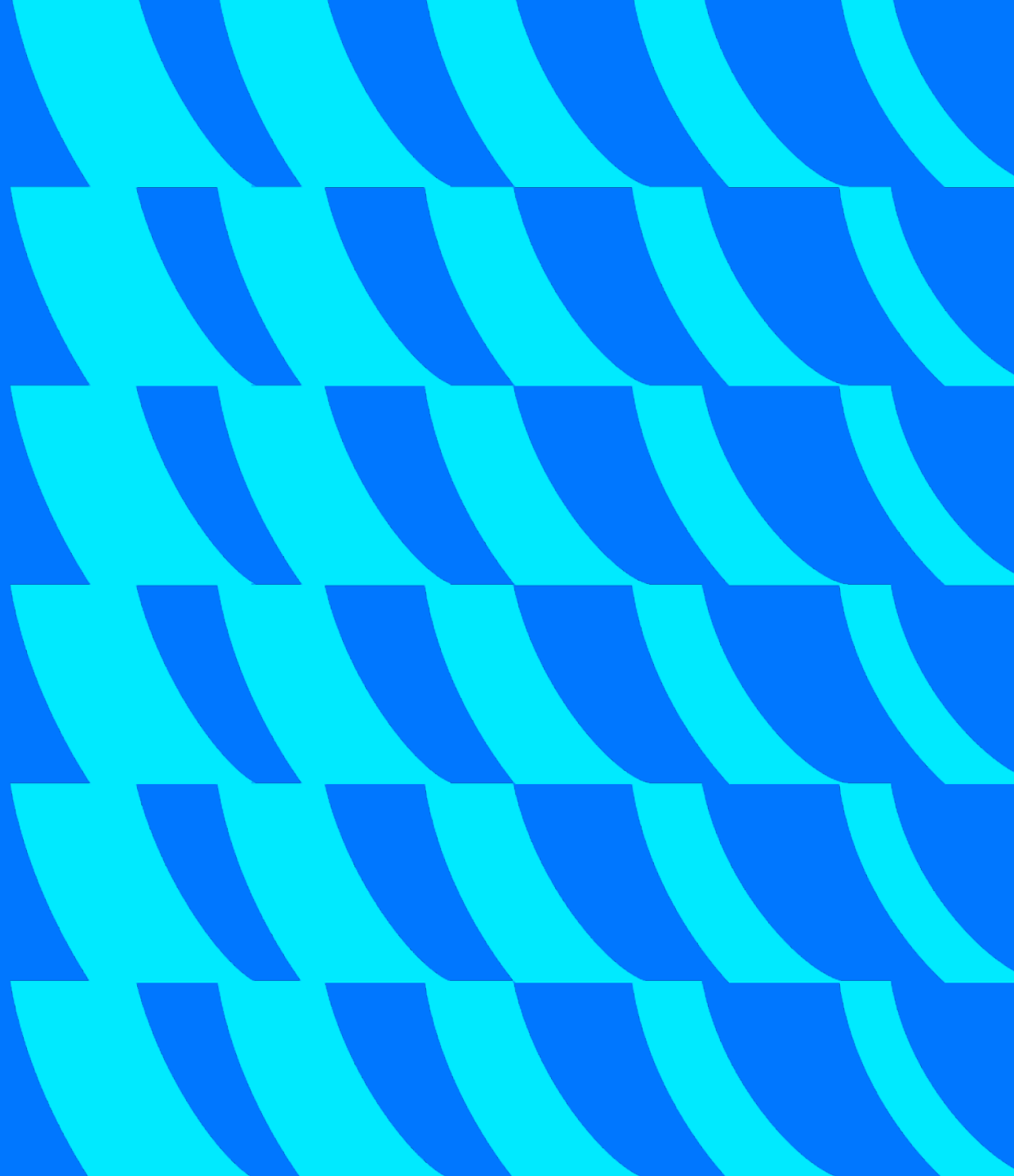
DataStore - Proto

```
object SettingsSerializer : Serializer<Settings> {  
    override val defaultValue: Settings = Settings.getDefaultInstance()  
  
    override suspend fun readFrom(input: InputStream): Settings {  
        try {  
            return Settings.parseFrom(input)  
        } catch (exception: InvalidProtocolBufferException) {  
            throw CorruptionException("Cannot read proto.", exception)  
        }  
    }  
  
    override suspend fun writeTo(  
        t: Settings,  
        output: OutputStream) = t.writeTo(output)  
    }  
  
val Context.settingsDataStore: DataStore<Settings> by datastore(  
    fileName = "settings.pb",  
    serializer = SettingsSerializer  
)
```


Feature	SharedPreferences	PreferencesDataStore	ProtoDataStore
Async API	✅ (only for reading changed values, via listener)	✅ (via Flow and RxJava 2 & 3 Flowable)	✅ (via Flow and RxJava 2 & 3 Flowable)
Synchronous API	✅ (but not safe to call on UI thread)	❌	❌
Safe to call on UI thread	❌(1)	✅ (work is moved to Dispatchers.IO under the hood)	✅ (work is moved to Dispatchers.IO under the hood)
Can signal errors	❌	✅	✅
Safe from runtime exceptions	❌(2)	✅	✅
Has a transactional API with strong consistency guarantees	❌	✅	✅
Handles data migration	❌	✅	✅
Type safety	❌	❌	✅ with Protocol Buffers

Вопросы?

Internal Storage



Internal Storage

Доступ к файлам в Internal Storage имеет только ваше приложение. Пользователь (в общем случае) доступа не имеет.

Чтение из хранилища:

```
FileInputStream openFileInput(String name)
```

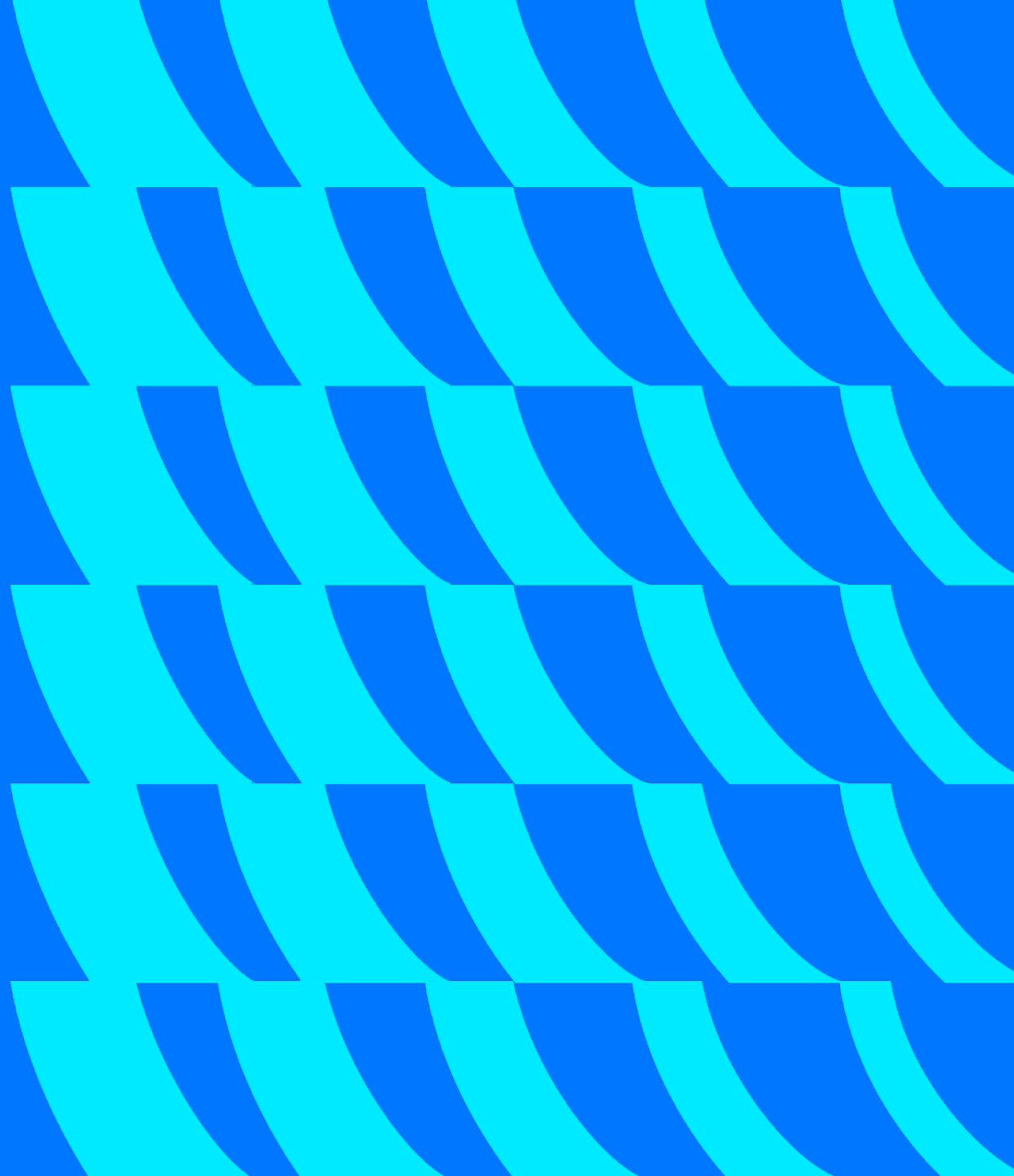
Запись в хранилище:

```
FileOutputStream openFileOutput(String name, int mode)
```

Internal Storage

- `File getFilesDir()` - путь до приватной директории приложения
- `File getDir(String name, int mode)` - открывает/создает директорию в приватном хранилище
- `File getCacheDir()` - путь до директории для хранения кэшей
- `boolean deleteFile(String name)` - удаляет приватный файл
- `String[] fileList()` - список приватных файлов

External Storage



External Storage

Добавить пермишены в `AndroidManifest.xml`:

- `WRITE_EXTERNAL_STORAGE` (с 29 API не всегда нужно!)
- `READ_EXTERNAL_STORAGE`

Проверка состояния внешнего хранилища:

```
Environment.getExternalStorageState()
```

Возможные состояния хранилища:

- `Environment.MEDIA_MOUNTED`
- `Environment.MEDIA_MOUNTED_READ_ONLY`

External Storage

```
fun isExternalStorageWritable: Boolean () {  
  
    val state = Environment.getExternalStorageState()  
  
    return Environment.MEDIA_MOUNTED == state  
  
}  
  
fun isExternalStorageReadable: Boolean () {  
  
    val state = Environment.getExternalStorageState()  
  
    return Environment.MEDIA_MOUNTED.equals(state) ||  
  
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));  
  
}
```


Общедоступные директории

Android сканирует некоторые директории с целью предоставить пользователю удобный доступ к ним. Получить путь до них можно с помощью:

```
File getExternalStoragePublicDirectory (String type)
```

Типы директорий:

- DIRECTORY_MUSIC
- DIRECTORY_PICTURES
- DIRECTORY_DOWNLOADS
- DIRECTORY_RINGTONES
- ...

Scoped Storage

С Android 11 permission `WRITE_EXTERNAL_STORAGE` не является необходимым и достаточным условием для записи на карту.

<https://developer.android.com/about/versions/11/privacy/storage>


<https://developer.android.com/training/data-storage/use-cases#share-media-all>

<https://developer.android.com/training/data-storage/manage-all-files>

Варианты:

1. Общедоступные коллекции - `MediaStore.Downloads`
2. `MediaStore.API`
3. В случае файлового менеджера - `Manage All Files`

Общедоступные директории



```
fun getAlbumStorageDir(albumName: String): File
{
    // Get the directory for the user's public pictures directory.
    val file = File(
        Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES
        ), albumName
    );
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

“Приватные” директории

Как Internal Storage для вашего приложения, но на внешнем носителе:

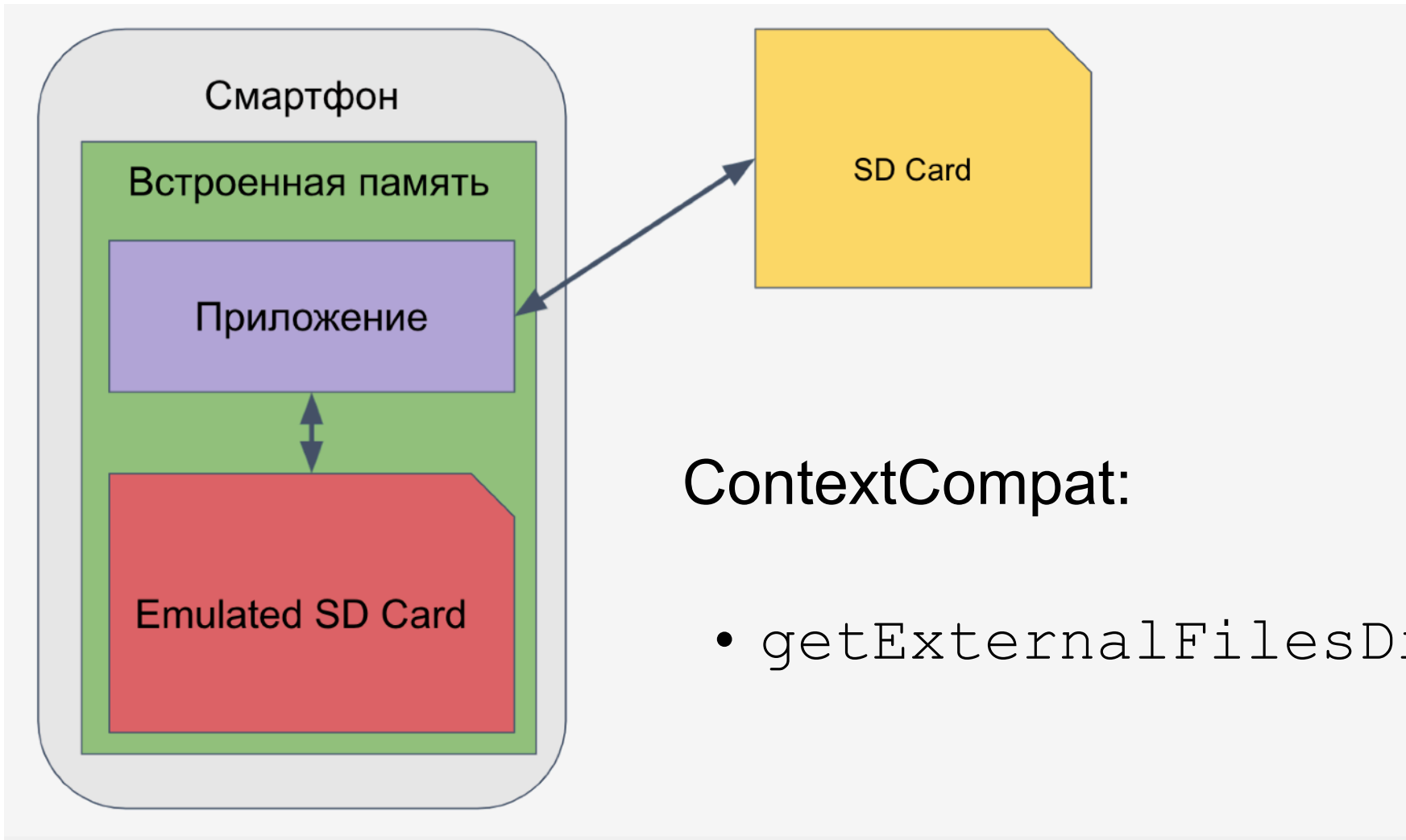
- `File getExternalFilesDir(String type)` – путь до приватной директории приложения
- `File getExternalCacheDir()` - путь до директории для хранения кэшей

`<uses-permission`

`android:name="android.permission.WRITE_EXTERNAL_STORAGE"`

`android:maxSdkVersion="28"`

Внешнее хранилище, но не внешнее



ContextCompat:

- `getExternalFilesDirs`

Куда устанавливается приложение?

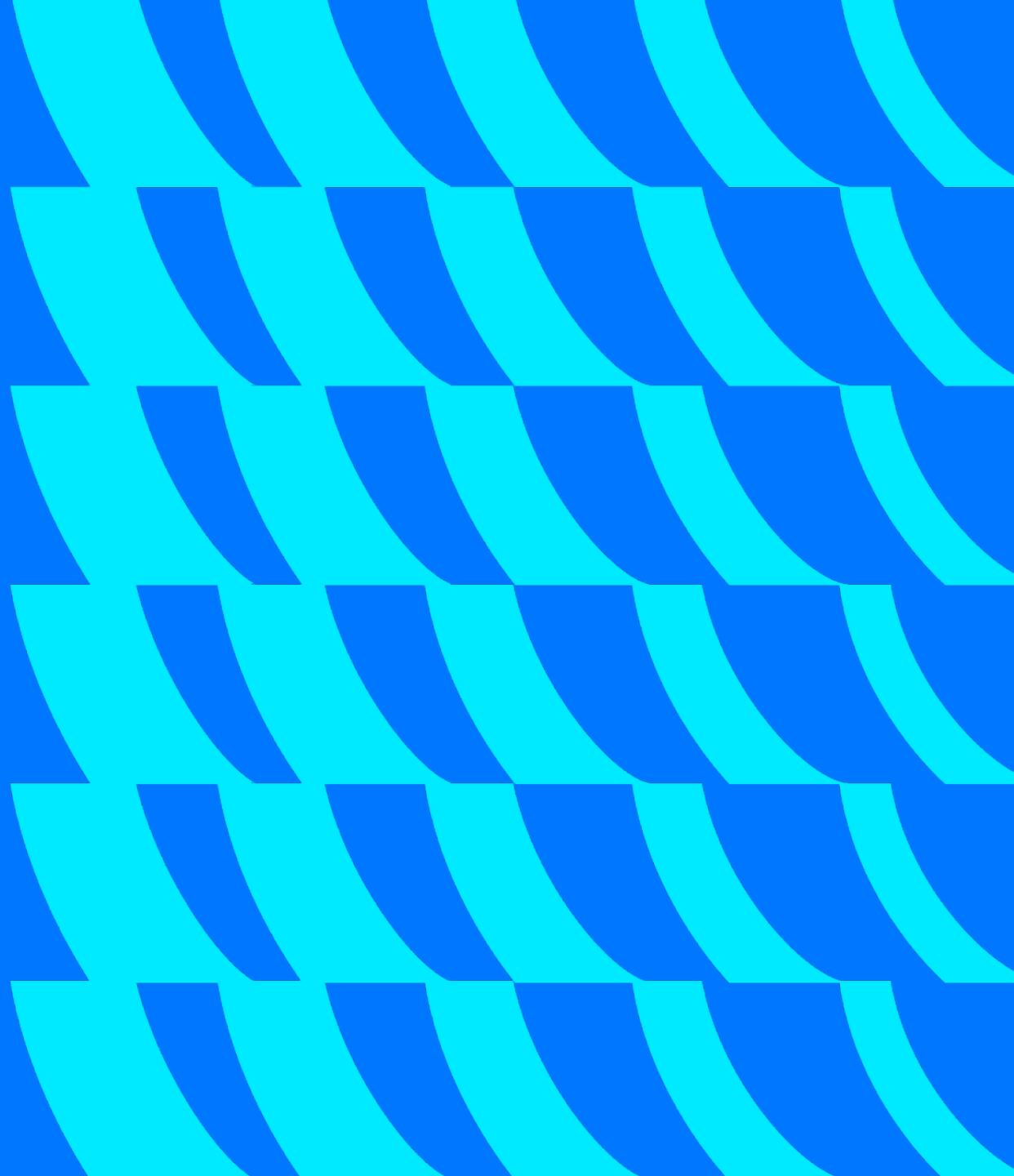
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
  
    android:installLocation="preferExternal"  
... >
```

Возможные варианты для `installLocation`:

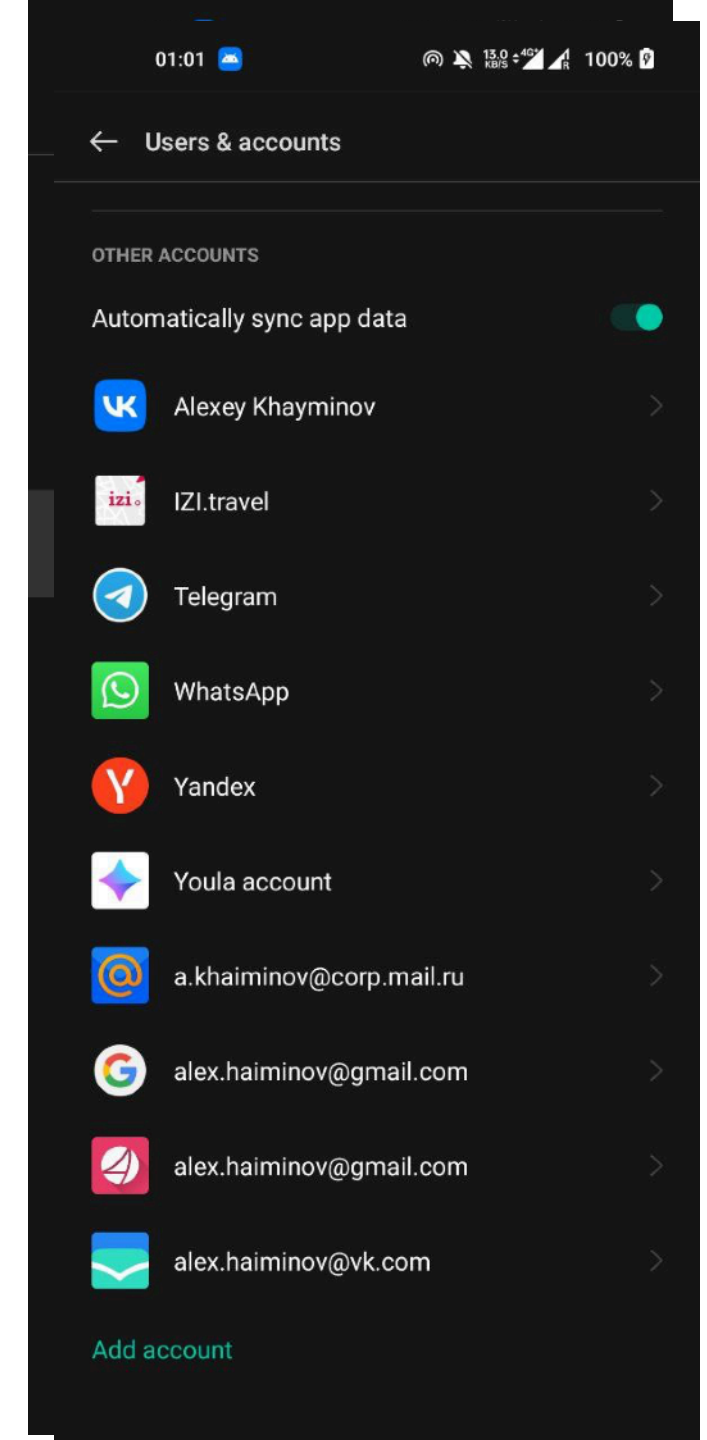
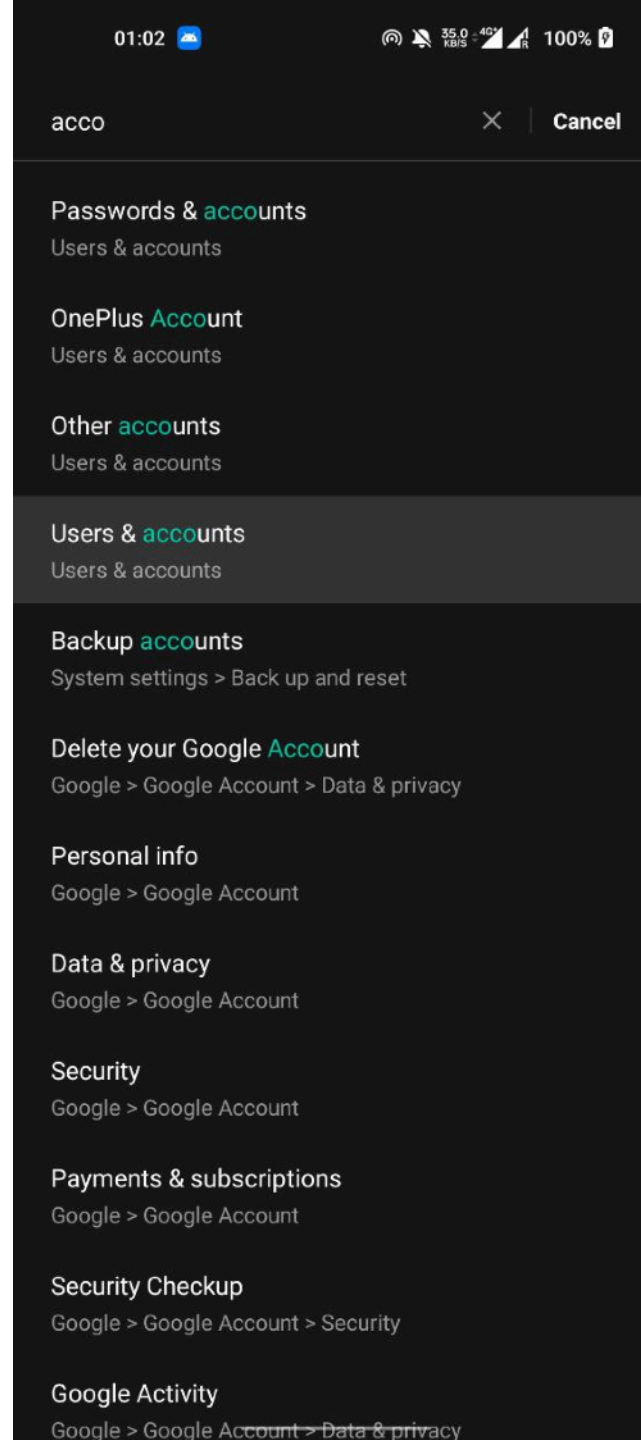
- `internalOnly`

Вопросы?

Account Manager



Accounts - where



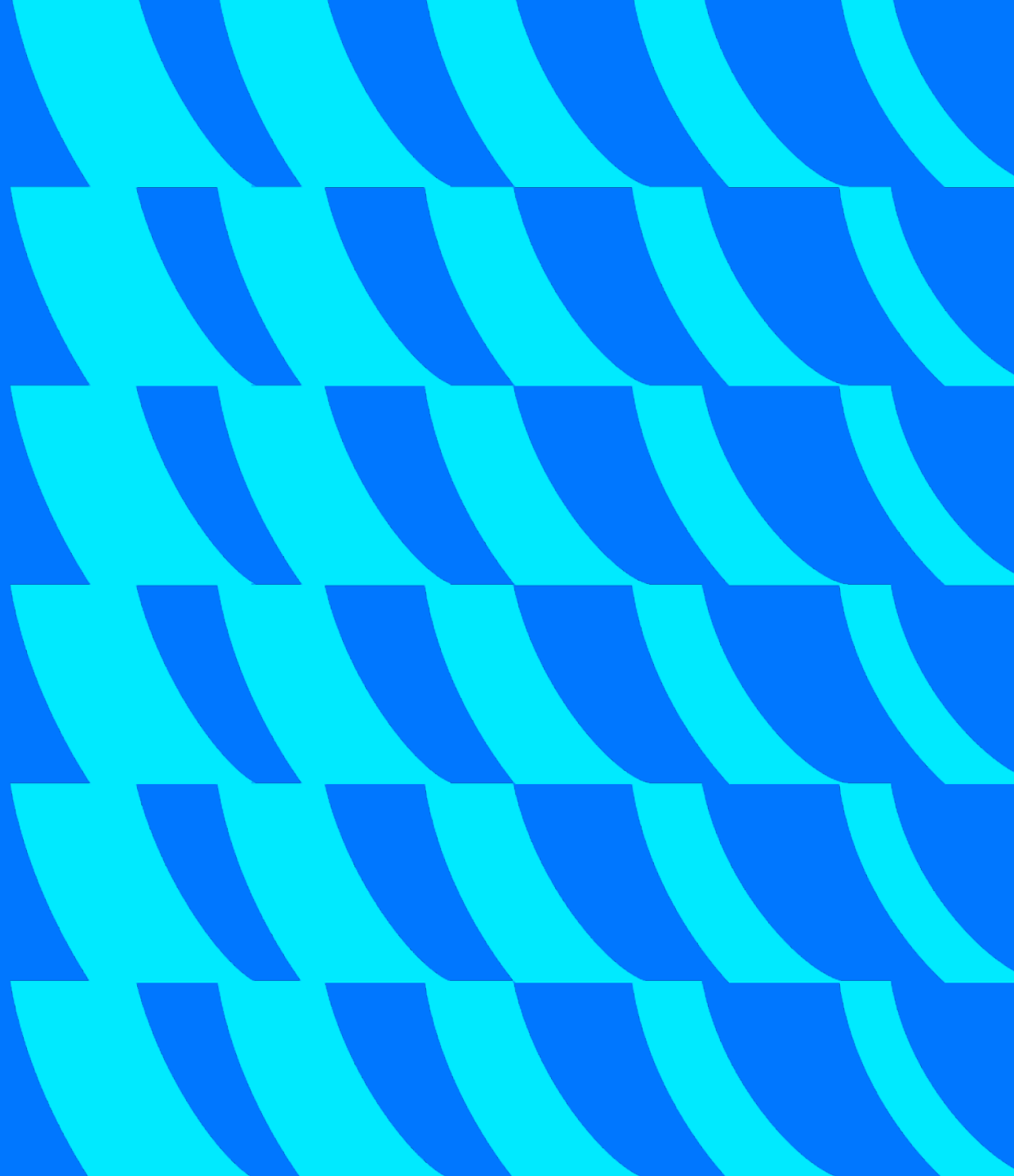
AccountCache

Permission - [Manifest.permission.GET_ACCOUNTS](#)

Распространённые методы:

- public **Account**[] getAccountsByType (**String** type)
- public [AccountManagerFuture](#)<[Bundle](#)> getAuthToken ([Account](#) account, **String** authTokenType, **Bundle** options, **Activity** activity, [AccountManagerCallback](#)<[Bundle](#)> callback, **Handler** handler)
- public **String** getPassword (**Account** account)
- public **String** getUserData (**Account** account, **String** key)

SQLite



SQLiteDatabase

ОСНОВНЫЕ МЕТОДЫ -

1. insert
2. delete
3. update
4. query
5. execSQL
6. rawQuery
7. beginTransaction/beginTransaction

SQLiteOpenHelper

```
class DbHelper(  
    context: Context?,  
    ) : SQLiteOpenHelper(context, DB_NAME, null, DB_VERSION) {  
    companion object {  
        const val DB_VERSION = 1  
        const val DB_NAME = "MY_DB"  
    }  
  
    override fun onCreate(db: SQLiteDatabase?) {  
        TODO("Not yet implemented")  
    }  
  
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
        TODO("Not yet implemented")  
    }  
  
    override fun onDowngrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
        TODO("Not yet implemented")  
    }  
}  
  
val db = DbHelper(this).readableDatabase.query(...)
```

SQLiteDatabase. Insert


```
private fun insertInternal(text: String) {  
    database = helper.writableDatabase  
    val contentValues = ContentValues()  
    contentValues.put(TEXT_COLUMN, text)  
    val insert = database.insert(TABLE_NAME, nullColumnHack: null, contentValues)  
}
```

SQLiteDatabase. Delete



```
val selection: String = COLUMN_NAME_TITLE.toString() + " LIKE ?"  
val selectionArgs = arrayOf("MyTitle")  
  
val affectedRows = db.delete(TABLE_NAME, selection, selectionArgs)
```

SQLiteDatabase. Update



```
val values = ContentValues()  
    values.put(COLUMN_NAME_TITLE, title)  
  
val selection: String = COLUMN_NAME_TITLE.toString() + " LIKE ?"  
val selectionArgs = arrayOf("MyTitle")  
  
val count = db.update(  
    TABLE_NAME,  
    values,  
    selection,  
    selectionArgs  
)
```


SQLiteDatabase. Query

```
val projection = arrayOf<String>(
    COLUMN_NAME_TITLE,
    COLUMN_NAME_SUBTITLE
)

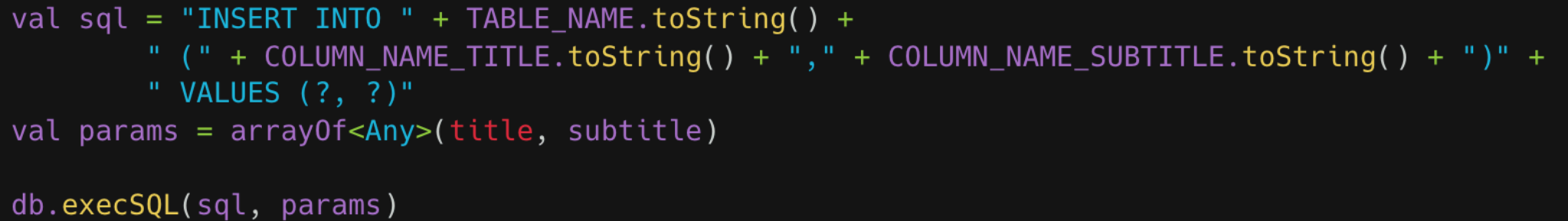
val selection: String = COLUMN_NAME_TITLE.toString() + " = ?"
val selectionArgs = arrayOf("My Title")

val sortOrder: String = COLUMN_NAME_SUBTITLE.toString() + " DESC"

val c: Cursor = db.query(
    TABLE_NAME,
    projection,
    selection,
    selectionArgs,
    null /*groupBy*/,
    null /*having*/,
    sortOrder
)
```

SQLiteDatabase. execSQL


Используем, если НЕ нужно вернуть данные. Например CREATE, DROP, PRAGMA и тд



```
val sql = "INSERT INTO " + TABLE_NAME.toString() +  
    " (" + COLUMN_NAME_TITLE.toString() + "," + COLUMN_NAME_SUBTITLE.toString() + ")" +  
    " VALUES (?, ?)"  
val params = arrayOf<Any>(title, subtitle)  
  
db.execSQL(sql, params)
```

SQLiteDatabase.rawQuery


rawQuery может вернуть данные. Используется для сложных запросов



```
val query = "SELECT " +  
    COLUMN_NAME_TITLE.toString() + "," + COLUMN_NAME_SUBTITLE.toString() +  
    " FROM " + TABLE_NAME.toString() + " WHERE " + COLUMN_NAME_TITLE.toString() + "=?"  
  
val params = arrayOf(title)  
  
val cursor = db.rawQuery(query, params)
```

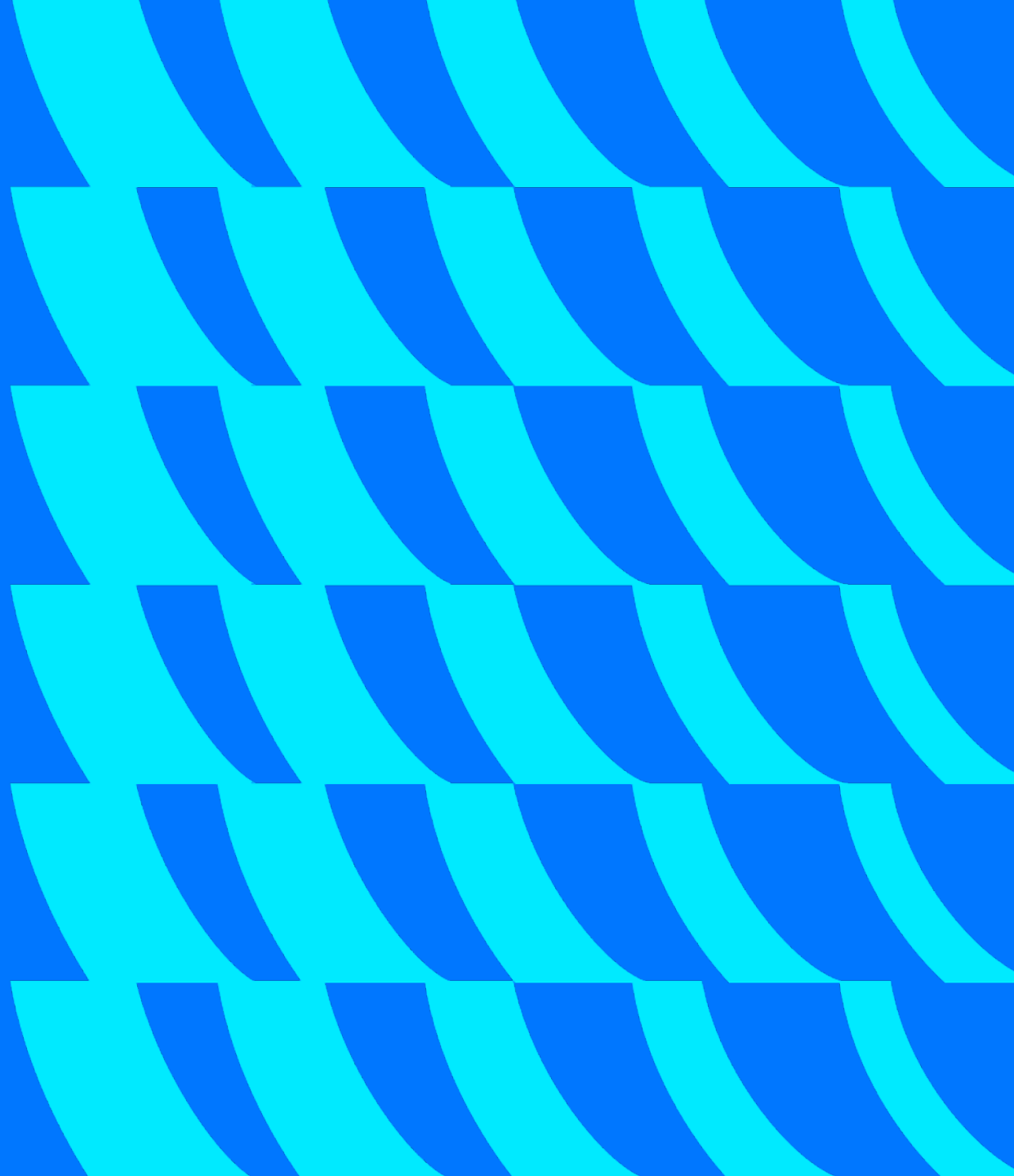
SQLiteDatabase. Transactions

Must-have фича. Позволяет кардинально ускорять батч-запросы и производить откаты в случае неуспешных операций



```
db.beginTransaction()  
try {  
    //select, insert, update, delete...  
    db.setTransactionSuccessful()  
} finally {  
    db.endTransaction()  
}
```

Content
Provider

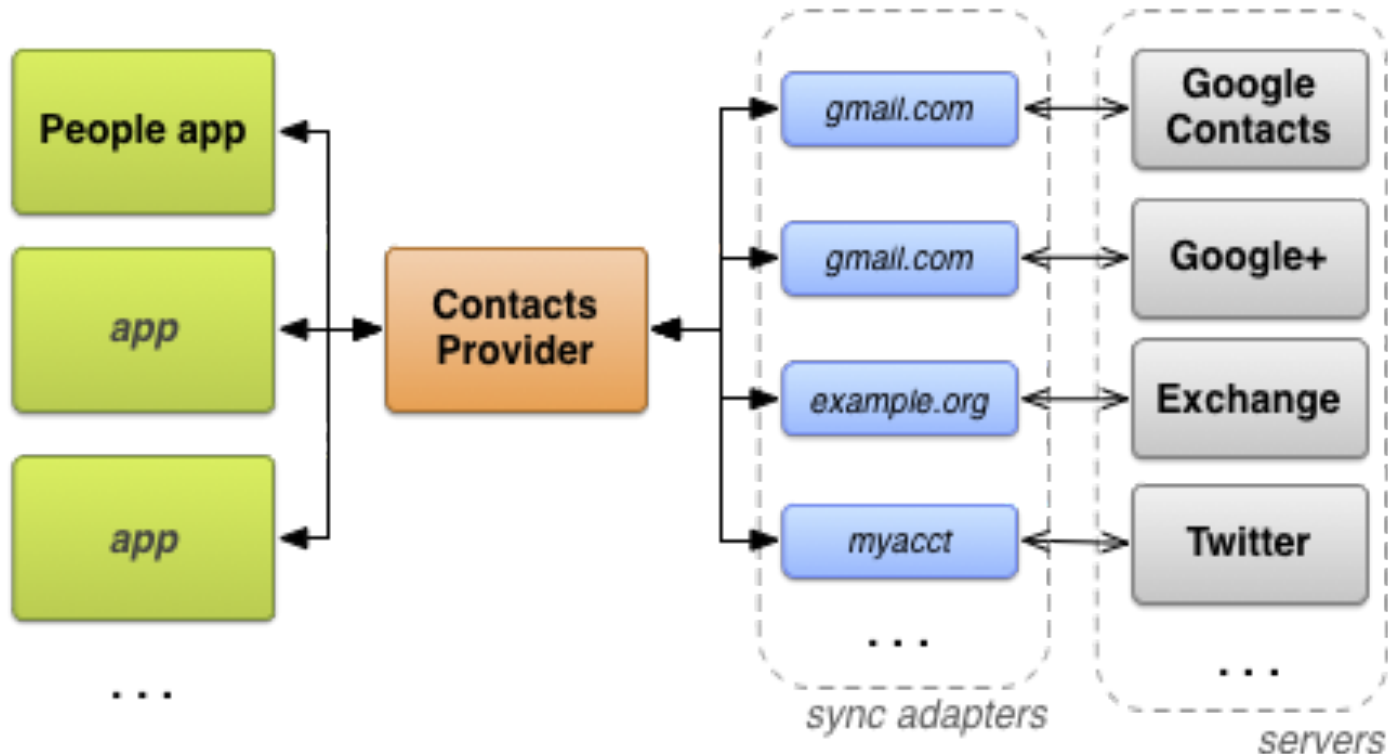


Content Provider

Нужен, чтобы обмениваться данными с другими приложениями

Самый яркий пример - контактная книга

Важно - можно реализовать CP на любом источнике данных - базе данных, текстовых файлах, зеркало удалённого источника



ContentProvider

ОСНОВНЫЕ МЕТОДЫ -

1. insert
2. delete
3. update
4. query
5. bulkInsert
6. setReadPermission/setWritePermission
7. getCalling<...> Package/Context/etc

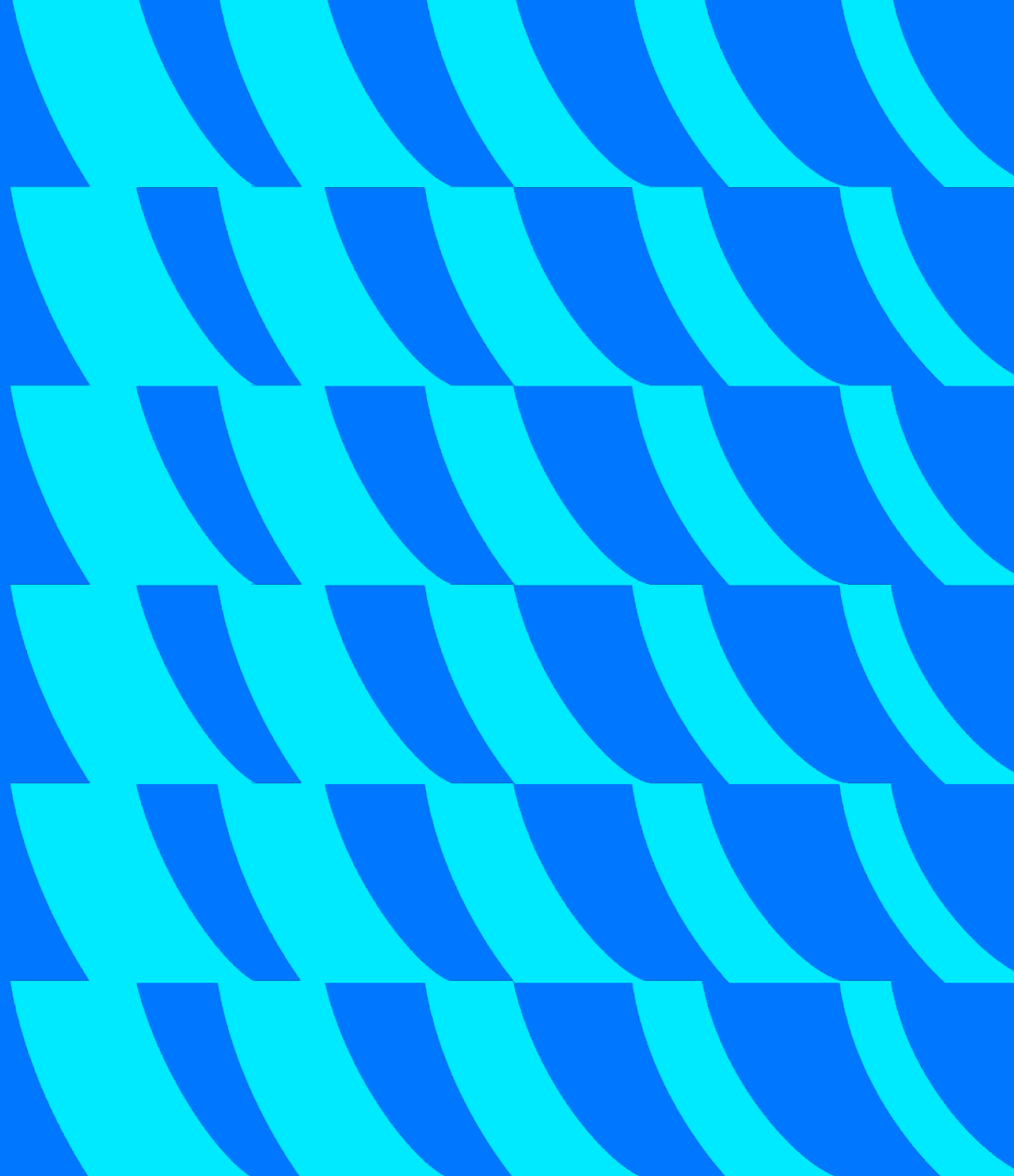
Content Provider

```
override fun onCreate(): Boolean {  
    database = DbManagerSqlite.getInstance(context!!).initialize()  
    //make uri matcher work  
    uriMatcher.addURI(DbUri.AUTHORITY, DbUri.ids, IDS.IDS)  
    uriMatcher.addURI(DbUri.AUTHORITY, DbUri.texts, IDS.TEXTS)  
    uriMatcher.addURI(DbUri.AUTHORITY, DbUri.text(id: "*"), IDS.TEXT)  
    return true  
}
```


Content Provider


```
override fun delete(uri: Uri, selection: String?, selectionArgs: Array<String>?): Int {  
    val count: Int = when (uriMatcher.match(uri)) {  
        IDS.IDS -> database.delete(DbManagerSqlIte.TABLE_NAME, selection, selectionArgs)  
        IDS.TEXTS -> database.delete(DbManagerSqlIte.TABLE_NAME, selection, selectionArgs)  
        IDS.TEXT -> database.delete(DbManagerSqlIte.TABLE_NAME, selection, selectionArgs)  
        else -> 0  
    }  
    //notify of changing  
    if (count > 0)  
        context!!.contentResolver.notifyChange(uri, observer: null)  
    return count  
}
```

Room



Room

Room — это высокоуровневый интерфейс для низкоуровневых привязок SQLite, встроенных в Android, о которых вы можете узнать больше в документации. Он выполняет большую часть своей работы во время компиляции, создавая API-интерфейс поверх встроенного SQLite API, поэтому вам не нужно работать с Cursor



```
@Entity
class Person {
    @PrimaryKey val name: String
    val age: Int
    favoriteColor: String
}
```

Room. DAO

```
@Dao
interface PersonDao {

    // Добавление Person в бд
    @Insert
    fun insertAll(vararg people: Person)

    // Удаление Person из бд
    @Delete
    fun delete(person: Person)

    // Получение всех Person из бд
    @Query("SELECT * FROM person")
    fun getAllPeople(): List<Person>

    // Получение всех Person из бд с условием
    @Query("SELECT * FROM person WHERE favoriteColor LIKE :color")
    fun getAllPeopleWithFavoriteColor(String color): List<Person>

}
```

Room. Create DB



```
@Database(entities =  
{Person::class.java /*, AnotherEntityType::class.java */}, version = 1)  
abstract class AppDatabase : RoomDatabase {  
    abstract fun getPersonDao(): PersonDao  
}  
  
val db = Room.databaseBuilder(getApplicationContext(),  
    AppDatabase::class.java, "populus-database").build()  
  
val everyone = db.getPersonDao().getAllPeople()
```

References

<https://developer.android.com/guide/topics/data/data-storage.html>

<https://developer.android.com/guide/topics/data/install-location.html>

<https://behr.com.ru/post/226106/>

Оставьте
отзыв!



Спасибо за внимание!

Хайминов Алексей