

5.2. Общее хранилище файлов

Сайт: [Samsung Innovation Campus](https://innovationcampus.ru)
Курс: Мобильная разработка на Kotlin
Книга: 5.2. Общее хранилище файлов

Напечатано.: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:59

Оглавление

- 1. 5.2.1. Обзор общего хранилища
- 2. 5.2.2. Доступ к медиафайлам из общего хранилища
- 3. 5.2.3. Доступ к документам и другим файлам из общего хранилища
- 4. 5.2.4. Сохранение пар "ключ-значение"
- 5. 5.2.4. Упражнение 5.2.

1. 5.2.1. Обзор общего хранилища

В случаях когда данные приложения должны быть доступны другим приложениям можно использовать общее хранилище пользовательских данных. Даже если пользователь удаляет ваше приложения, информация в общем хранилище сохраняется. Само название указывает на то, что его можно использовать в приложении на устройстве. Любое приложение может получить доступ к этим данным. До версии Android Q разработчик должен был получать разрешения от пользователя на чтение и запись данных в общее хранилище. Начиная с Android Q появилось [scoped storage](#) и теперь разработчику не нужно получать разрешение на запись, чтение и изменение собственных данных нашего приложения. Scoped Storage — это изолированные куски памяти, выделяемые каждому приложению, впрочем, это даже можно понять по названию. В настоящее время в Android приложения работают с памятью по аналогии с Windows — они получают доступ ко всей файловой системе. Scoped Storage позволит изолировать определенные ячейки памяти под конкретные приложения. Есть несколько ключевых причин такого нововведения. С одной стороны использование устройств становится более безопасным, с другой стороны, в файловой системе устройства стало больше детерминированности.

Благодаря концепции хранилища Scoped Android система знает, какой файл создается каким приложением, и за счет чего происходит лучшее управления файлами приложения. Из Android Q и выше разработчик не может напрямую получить доступ к расположению общих данных. Для доступа к ним необходимо указать в манифесте приложения разрешение «ACCESS_MEDIA_LOCATION».

Android предоставляет API для хранения и доступа к следующим типам общих данных:

- **Медиа.** В Android есть общедоступные каталоги для медиаресурсов: фото, аудиофайлы и т.д. Приложение может получить доступ к медиаконтенту с использованием [MediaStore API](#) платформы, о котором речь пойдет позже.
- **Документы.** В системе есть специальный каталог для хранения документов (например файлов PDF или книг использующие в формате EPUB). Для доступа к этим файлам можно использовать [Storage Access Framework](#).
- **Датасеты.** На Android 11 (уровень API 30) и выше система кэширует большие наборы данных, используемые несколькими приложениями. Данные наборы могут использоваться например при решении задач машинного обучения. Приложения могут получить доступ к этим общим наборам данных с помощью [API BlobStoreManager](#).

2. 5.2.2. Доступ к медиафайлам из общего хранилища

[MediaStore API](#) - это рекомендуемый способ работы с медиафайлами (изображения, аудио, видео). Для взаимодействия с хранилищем мультимедиа файлов необходимо использовать объект [ContentResolver](#), извлекаемый из контекста приложения.

MediaStore API сохраняет данные в виде базы данных. Для каждого вида файла существует отдельная таблица для хранения данных. Для хранения обычно используется три типа файлов: изображения, видео и аудио. Начиная с версии Android 10, добавлена новая коллекция Download Collections. Для изображений используют [MediaStore.Images](#), для видео - таблица [MediaStore.Video](#), для аудиофайлов - [MediaStore.Audio](#), для загрузок - таблица [MediaStore.Downloads](#).

В MediaStore API для чтения данных с устройства, предоставленных приложением, нет необходимости получать специальное разрешение от пользователя. Однако, для обработки данных, которые не создавались нашим приложением, необходимо получить соответствующие разрешения от пользователя устройства на доступ к ним.

Доступ к данным с помощью Media Store API

Получить файлы мультимедиа можно следующим способом:

```
val projection = arrayOf(MediaStore.Images.Media._ID, MediaStore.Images.Media.DISPLAYNAME, MediaStore.Images.Media.DATE_TAKEN)

val selection = "${MediaStore.Images.Media.DATE_TAKEN} >= ?"

val selectionArgs = arrayOf dateToTimestamp(day = 24, month = 7, year = 2019).toString()

val sortOrder = "${MediaStore.Images.Media.DATE_TAKEN} DESC"

getApplication().contentResolver.query( MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
projection,
selection,
selectionArgs,
sortOrder)?.use
{ cursor -> imageUrl = addImagesFromCursor(cursor) }
```

В коде выше:

- **projection** - массив, содержащий всю необходимую информацию медиафайла. Это похоже на запрос select базы данных.
- **selection** - аналогично оператору where при запросах в базе данных. Содержит условие, на основании которого должны быть получены данные.
- **selectionArgs** - массив, содержащий значения, по которым осуществляется выбор.
- **sortOrder** - ключ, используемый для сортировки данных по столбцу и порядку. По умолчанию порядок возрастающий. Для убывающего порядка, используйте ключевое слово DESC.
- **query()** - метод класса **ContentResolver**, который принимает все вышеперечисленные параметры, а также дополнительный параметр **Uri**, который сопоставляется с требуемой таблицей в провайдера.

Таким образом, чтобы найти медиа файл, удовлетворяющий определенному набору условий, например, продолжительностью 5 минут или более, используйте SQL-подобную инструкцию выбора, аналогичную той, которая показана в следующем фрагменте кода:

```
// Требуется разрешение READ_EXTERNAL_STORAGE для доступа к видеофайлам, которые ваше приложение не создавало.

// Контейнер для информации о видео
data class Video(val uri: Uri,
    val name: String,
    val duration: Int,
    val size: Int
)
val videoList = mutableListOf<Video>()

val collection =
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        MediaStore.Video.Media.getContentUri(
            MediaStore.VOLUME_EXTERNAL
        )
    } else {
        MediaStore.Video.Media.EXTERNAL_CONTENT_URI
    }

val projection = arrayOf(
    MediaStore.Video.Media._ID,
    MediaStore.Video.Media.DISPLAY_NAME,
    MediaStore.Video.Media.DURATION,
    MediaStore.Video.Media.SIZE
)

// Показывать только видео продолжительностью не менее 5 минут
val selection = "${MediaStore.Video.Media.DURATION} >= ?"
val selectionArgs = arrayOf(
    TimeUnit.MILLISECONDS.convert(5, TimeUnit.MINUTES).toString()
)

// Отображайте видео в алфавитном порядке в зависимости от их отображаемого имени
val sortOrder = "${MediaStore.Video.Media.DISPLAY_NAME} ASC"

val query = ContentResolver.query(
    collection,
    projection,
    selection,
    selectionArgs,
    sortOrder
)
query?.use { cursor ->
    // Индексы столбцов кеширования
    val idColumn = cursor.getColumnIndexOrThrow(MediaStore.Video.Media._ID)
    val nameColumn =
        cursor.getColumnIndexOrThrow(MediaStore.Video.Media.DISPLAY_NAME)
    val durationColumn =
        cursor.getColumnIndexOrThrow(MediaStore.Video.Media.DURATION)
    val sizeColumn = cursor.getColumnIndexOrThrow(MediaStore.Video.Media.SIZE)

    while (cursor.moveToNext()) {
        // Получить значения столбцов для данного видео
        val id = cursor.getLong(idColumn)
        val name = cursor.getString(nameColumn)
        val duration = cursor.getInt(durationColumn)
        val size = cursor.getInt(sizeColumn)

        val contentUri: Uri = ContentUris.withAppendedId(
            MediaStore.Video.Media.EXTERNAL_CONTENT_URI,
            id
        )

        // сохраняю значения столбцов и contentUri в локальном объекте, представляющем медиафайл.
        videoList += Video(contentUri, name, duration, size)
    }
}
```

Если ваше приложение показывает несколько медиафайлов и просит пользователя выбрать один из них, более эффективно загружать предварительные версии или эскизы файлов вместо самих файлов. Это можно сделать следующим образом:

```
val thumbnail: Bitmap =
    applicationContext.contentResolver.loadThumbnail(
        content-uri, Size(640, 480), null)
```

Открытие медиафайла

При открытии медиафайлов можно использовать несколько вариантов. Логика их использования во многом зависит от того как лучше представлять медиаконтент: в виде файлового дескриптора, потока файлов или пути к файлу. Чтобы открыть медиафайл с помощью файлового дескриптора, необходимо задействовать логику, аналогичную следующей:

```
val resolver = applicationContext.contentResolver
val readOnlyMode = "r"
resolver.openFileDescriptor(content-uri, readOnlyMode).use { pfd ->
    // операции с pdf
}
```

Открыть медиафайл с помощью файлового потока можно например следующим образом:

```
val resolver = applicationContext.contentResolver
resolver.openInputStream(content-uri).use { stream ->
    // операции с файловым потоком
}
```

Более подробно тема доступа к медиафайлам и соображения при доступе к медиа-контенту представлены в [официальной документации](#)

Сохранение данных

Далее приведен пример сохранения данных с помощью Scoped Storage:

```
val values = ContentValues().apply {
    put(MediaStore.Images.Media.DISPLAY_NAME, name)
    put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg")
    put(MediaStore.Images.Media.RELATIVE_PATH, "Pictures/$bucketName/")
    put(MediaStore.Images.Media.IS_PENDING, 1)
}
val collection = MediaStore.Images.Media.getContentUri(MediaStore.VOLUME_EXTERNAL_PRIMARY)
val imageUri = context.contentResolver.insert(collection, values)
context.contentResolver.openOutputStream(imageUri).use { out ->
    bmp.compress(Bitmap.CompressFormat.JPEG, 90, out)
}
values.clear()
values.put(MediaStore.Images.Media.IS_PENDING, 0)
context.contentResolver.update(imageUri, values, null, null)
```

Чтобы удалить файл мультимедиа с помощью MediaStore API можно задействовать логику заложенную в следующем фрагменте кода:

```
try {
    getApplication().contentResolver.delete(
        image.contentUri, "${MediaStore.Images.Media._ID} = ?",
        arrayOf(image.id.toString()) )
}
catch (securityException: SecurityException) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        val recoverableSecurityException =
            securityException as? RecoverableSecurityException
        ?: throw securityException
        pendingDeleteImage = image
        _permissionNeededForDelete.postValue(
            recoverableSecurityException.userAction.actionIntent.intentSender
        )
    } else {
        throw securityException
    }
}
```

Чтобы вызвать метод `contentResolver.delete()`, его следует поместить в блок `try`, потому что он может вызвать исключение безопасности во время выполнения.

Начиная с Android R появились методы для получение массового доступа для изменения (`createWriteRequest`) и удаления (`createDeleteRequest`) данных. Например удаление можно осуществить следующим образом:

```
fun deleteMediaBulk(context: Context, media: List): IntentSender {
    val uris = media.map { it.uri }
    return MediaStore.createDeleteRequest(context.contentResolver, uris).intentSender
}
```

Завершая далеко не полный обзор MediaStore API следует отметить, что тема сохранения, получения доступа к медиафайлам, их удаления и т.д. довольно обширная. Для получения дополнительной информации о том, как хранить медиа файлы и получать к ним доступ, обратитесь к следующим [примерам](#) и [официальной документации](#).

3. 5.2.3. Доступ к документам и другим файлам из общего хранилища

На устройствах с Android 4.4 и выше ваше приложение может взаимодействовать с документами. Существует возможность выбора конкретных [провайдеров для документов](#) и других файлов для создания, открытия или изменения из приложения. Поскольку пользователь участвует в выборе файлов или каталогов, к которым может получить доступ приложение, каких-либо специальных системных разрешений не требуется. Кроме того, файлы, которые хранятся вне каталога приложения и вне хранилища мультимедиа, остаются на устройстве даже после его удаления.

Для работы с документами и другими файлами рекомендовано использовать фреймворк [Storage Access Framework \(SAF\)](#) (это просто стандартное приложения для выбора файлов). Прежде всего, мы должны подготовить Intent с конкретным действием, которое используется в методе `startActivityForResult`. При этом следует помнить об [ограничениях, введенных в Android 11](#). Позже, в `onActivityResult` мы ждем результата работы SAF, в котором мы получаем URI. Этот URI - это все, что нам нужно. Мы можем использовать его для создания правильного потока и выполнения операций с файлом (чтения из него или записи в него).

Storage Access Framework поддерживает следующие варианты использования для доступа к файлам и другим документам:

- **Создание нового файла.** При этом в намерения передается константа `ACTION_CREATE_DOCUMENT`. В данном варианте пользователям позволено сохранять файл в определенном месте.
- **Открытие файла или документа.** При этом в намерения передается константа `ACTION_OPEN_DOCUMENT`, что позволяет пользователям выбрать конкретный документ или файл для открытия.
- **Предоставление доступа к содержимому каталога.** Доступно с версии Android 5.0 и выше. При этом в намерения передается константа `ACTION_OPEN_DOCUMENT_TREE`. Позволяет пользователям выбирать определенный каталог, предоставляя приложению доступ ко всем его файлам и подкаталогам.

В следующем фрагменте кода показано, как создать и вызвать намерение для создания файла:

```
const val CREATE_FILE = 1

private fun createFile(pickerInitialUri: Uri) {
    val intent = Intent(Intent.ACTION_CREATE_DOCUMENT).apply {
        addCategory(Intent.CATEGORY_OPENABLE)
        type = "application/pdf"
        putExtra(Intent.EXTRA_TITLE, "invoice.pdf")

        // При желании укажите URI для каталога, который должен быть открыт в средстве выбора файлов системы,
        // прежде чем ваше приложение создаст документ.
        putExtra(DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
    }
    startActivityForResult(intent, CREATE_FILE)
}
```

После того, как пользователь выбрал каталог, нам все еще нужно обработать Uri результата в методе `onActivityResult`.

```
override fun onActivityResult(
    requestCode: Int, resultCode: Int, resultData: Intent?) {
    if (requestCode == CREATE_FILE && resultCode == Activity.RESULT_OK) {
        // Данные результата содержат URI для каталога, выбранного пользователем.
        resultData?.data?.also { uri ->
            // сохраните ваши данные с помощью `uri`
        }
    }
}
```

В следующем фрагменте кода показано, как создать и вызвать намерение для открытия документа PDF:

```
const val PICK_PDF_FILE = 2

fun openFile(pickerInitialUri: Uri) {
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {
        addCategory(Intent.CATEGORY_OPENABLE)
        type = "application/pdf"

        putExtra(DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
    }

    startActivityForResult(intent, PICK_PDF_FILE)
}
```

В следующем фрагменте кода показано, как создать и вызвать намерение для открытия каталога:

```
fun openDirectory(pickerInitialUri: Uri) {
    // Выберете каталог с помощью средства выбора файлов системы.
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT_TREE).apply {
        // Можно указать URI для каталога, который должен открываться в средстве выбора файлов при его загрузке.

        putExtra/DocumentsContract.EXTRA_INITIAL_URI, pickerInitialUri)
    }

    startActivityForResult(intent, your-request-code)
}
```

После того как пользователь выбрал файл или каталог с помощью средства выбора файлов системы, вы можете получить URI выбранного элемента, используя следующий код в onActivityResult():

```
override fun onActivityResult(
    requestCode: Int, resultCode: Int, resultData: Intent?) {
    if (requestCode == CREATE_FILE
        && resultCode == Activity.RESULT_OK) {
        // Данные результата содержат URI для документа или каталога, выбранного пользователем.
        resultData?.data?.also { uri ->
            // Выполняйте операции с документом, используя его URI.
        }
    }
}
```

Когда у вас есть URI для документа, вы получаете доступ к его метаданным, также вы можете открыть документ для дальнейшей обработки. В следующем фрагменте кода показано, как открыть файл растрового изображения с учетом его URI:

```
val contentResolver = applicationContext.contentResolver

@Throws(IOException::class)
private fun getBitmapFromUri(uri: Uri): Bitmap {
    val parcelFileDescriptor: ParcelFileDescriptor =
        contentResolver.openFileDescriptor(uri, "r")
    val fileDescriptor: FileDescriptor = parcelFileDescriptor.fileDescriptor
    val image: Bitmap = BitmapFactory.decodeFileDescriptor(fileDescriptor)
    parcelFileDescriptor.close()
    return image
}
```

Следующий фрагмент кода перезаписывает содержимое документа по URI:

```
val contentResolver = applicationContext.contentResolver

private fun alterDocument(uri: Uri) {
    try {
        contentResolver.openFileDescriptor(uri, "w")?.use {
            FileOutputStream(it.fileDescriptor).use {
                it.write(
                    ("Overwritten at ${System.currentTimeMillis()}\n")
                        .toByteArray()
                )
            }
        }
    } catch (e: FileNotFoundException) {
        e.printStackTrace()
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
```

Более подробно про доступ к документам и другим файлам из общего хранилища можно прочитать в [официальной документации](#)

4. 5.2.4. Сохранение пар "ключ-значение"

В данном параграфе рассмотрим способ хранения данных, основанных на хранении ассоциативных массивов — **Shared Preferences**. Класс **SharedPreferences** позволяет создавать в приложении именованные ассоциативные массивы типа «ключ — значение», которые могут быть использованы различными компонентами приложения.

Если есть небольшая коллекция из пар "ключ-значение", которые вы хотите сохранить, необходимо использовать [SharedPreferences API](#). Объект **SharedPreferences** указывает на файл, содержащий пары ключ-значение, и предоставляет простые методы для их чтения и записи.

Чтобы получить экземпляр класса **SharedPreferences** для получения доступа к настройкам в коде приложения используются следующие методы:

- **getPreferences()** — внутри активности, чтобы обратиться к определенному для активности предпочтению;
- **getSharedPreferences()** — внутри активности, чтобы обратиться к предпочтению на уровне приложения;

Например, следующий код обращается к файлу общих настроек, определяемому строкой ресурса **R.string.preference_file_key**, и открывает его в закрытом режиме, чтобы файл был доступен только вашему приложению:

```
val sharedPref = activity?.getSharedPreferences(  
    getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

Если вам нужен один общий файл настроек для вашей активности, вы можете использовать метод **getPreferences()**:

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```

В названии файлов общих настроек следует использовать имя, которое уникально для вашего приложения. Простой способ сделать это - добавить в имя файла идентификатор вашего приложения. Например следующим образом: "com.example.myapplication.PREFERENCE_FILE_KEY". Все модификаторы, кроме **MODE_PRIVATE**, в настоящий момент объявлены deprecated и не рекомендуются к использованию в целях безопасности.

Для записи данных в файл общих настроек необходимо создать объект **SharedPreferences.Editor** и вызвать метод **edit()**. Далее передайте ключи и значения, которые вы хотите записать в методы **putInt()** или **putString()**, а затем вызовите **apply()** или **commit()**, чтобы сохранить изменения. Следующий код демонстрирует запись пары "R.string.myString - newMyString":

```
val sharedPreferences = activity?.getPreferences(Context.MODE_PRIVATE) ?: return  
with (sharedPreferences.edit()) {  
    putInt(getString(R.string.myString), newMyString)  
    apply()  
}
```

Метода **apply()** сразу изменяет объект **SharedPreferences** в памяти, но записывает данные асинхронно. Для синхронной записи задействуйте метод **commit()** (следует избегать его вызова из основного потока, поскольку это может приостановить рендеринг пользовательского интерфейса).

Для чтения значений из файла общих настроек, требуется вызов методов **getInt()** и **getString()**. В качестве аргумента необходимо передать ключ для извлечения нужного значения и, возможно, значение по умолчанию, которое будет возвращено, если ключ отсутствует. Сделать это можно следующим образом:

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return  
val defaultValue = resources.getInteger(R.integer.my_int_default_key)  
val myValue = sharedPref.getInt(getString(R.string.my_saved_int_value_key), defaultValue)
```

5. 5.2.4. Упражнение 5.2.

Разработаем приложение, в котором по нажатию кнопки в текстовом поле будут отображаться имена файлов двадцати последних сохраненных на устройстве изображений вместе с датой их последнего изменения.

Приложение будет состоять из одной активности, на которой будет изображена кнопка для загрузки названий файлов с датами из общего хранилища изображений, а также `ScrollView` с дочерним элементом `TextView` для отображения информации. Разметка `activity_main.xml` может быть следующей:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnImages"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="Load Images"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/btnImages"
        android:fillViewport="true">

        <TextView
            android:id="@+id/textResults"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="8dp" />
    </ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

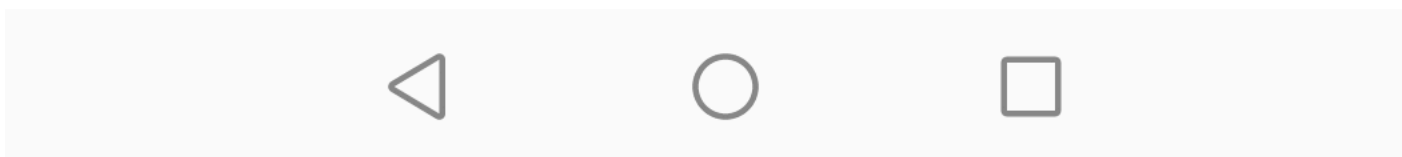
MTS RUS



31 % 22:04

MediaStoreImageExample

LOAD IMAGES



Для получения разрешения на чтение файлов из общего хранилища в манифест приложения необходимо добавить следующую строчку:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

С соответствии с функционалом приложения в классе главной активности объявим текстовое поле, свойство для хранения даты изменения изображения. В методе `onCreate` инициализируем текстовое представление для вывода информации и установим слушателя на кнопку:

```
lateinit var textResults: TextView
private val dateFormat = SimpleDateFormat.getInstance()

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    textResults = findViewById(R.id.textResults)

    findViewById<Button>(R.id.btnImages).setOnClickListener {
        loadImages()
    }
}
```

Далее реализуем метод `loadImages()`.

```
private fun loadImages() {
    if (ContextCompat.checkSelfPermission(
        this,
        Manifest.permission.READ_EXTERNAL_STORAGE
    ) == PackageManager.PERMISSION_GRANTED
    ) {
        fetchImagesAndShowResult()
    } else {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
            REQ_IMAGES
        )
    }
}
```

В методе происходит проверка доступа к чтению файлов из общего хранилища. Чтобы проверить, есть ли разрешение, вызывается метод `ContextCompat.checkSelfPermission()`. Иначе с помощью метода `activitycompat.requestPermission()` запрашивается разрешение на использование соответствующих разрешений.

Далее реализуем метод обратного вызова `onRequestPermissionsResult()`, который вызывается для получения результата запроса после разрешения пользователем приложению использовать общее хранилище.

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (ContextCompat.checkSelfPermission(
        this,
        Manifest.permission.READ_EXTERNAL_STORAGE
    ) == PackageManager.PERMISSION_GRANTED
    ) {
        fetchImagesAndShowResult()
    } else {
        Snackbar.make(
            findViewById(android.R.id.content),
            "Can't get data without permission",
            Snackbar.LENGTH_SHORT
        ).show()
    }
}
```

Далее необходимо реализовать метод `fetchImagesAndShowResult()`:

```
private fun fetchImagesAndShowResult() {
    val stringBuilder = StringBuilder()

    val projection = arrayOf( // media-database-columns-to-retrieve
        MediaStore.Images.ImageColumns.DISPLAY_NAME,
        MediaStore.Images.ImageColumns.DATE_MODIFIED
    )

    val selection = null // sql-where-clause-with-placeholder-variables
    val selectionArgs = null // values-of-placeholder-variables
    val sortOrder =
        "${MediaStore.Images.ImageColumns.DATE_MODIFIED} DESC LIMIT 20" // sql-order-by-clause

    applicationContext.contentResolver.query(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        projection,
        selection,
        selectionArgs,
        sortOrder
   )?.use { cursor ->
        val nameColumn =
            cursor.getColumnIndexOrThrow(MediaStore.Images.ImageColumns.DISPLAY_NAME)
        val dateModifiedColumn =
            cursor.getColumnIndexOrThrow(MediaStore.Images.ImageColumns.DATE_MODIFIED)

        while (cursor.moveToNext()) {
            stringBuilder.append(cursor.getString(nameColumn)).append("\n")
                .append(dateFormat.format(cursor.getLong(dateModifiedColumn) * 1000L))
                .append("\n\n")
        }
        textResults.text = stringBuilder
    }
}
```

Чтобы взаимодействовать с хранилищем мультимедиа, используется объект `ContentResolver`. Система автоматически сканирует внешнее хранилище изображений, используя Media Store API. Чтобы найти файлы, удовлетворяющие определенному набору условий (в нашем случае установлен лимит в 20 файлов) используется SQL-подобный оператор выбора.

Полностью класс `MainActivity.kt` будет выглядеть следующим образом:

```
package ru.samsung.itacademy.mdev.mediastoreimageexample

import android.Manifest
import android.content.pm.PackageManager
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Button
import android.widget.TextView
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.android.material.snackbar.Snackbar
import java.text.SimpleDateFormat

class MainActivity : AppCompatActivity() {
    lateinit var textResults: TextView
    private val dateFormat = SimpleDateFormat.getInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        textResults = findViewById(R.id.textResults)

        findViewById<Button>(R.id.btnImages).setOnClickListener {
            loadImages()
        }

        private fun loadImages() {
            if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) == PackageManager.PERMISSION_GRANTED
            ) {
                fetchImagesAndShowResult()
            } else {
                ActivityCompat.requestPermissions(
                    this,
                    arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
                    REQ_IMAGES
                )
            }
        }

        override fun onRequestPermissionsResult(
            requestCode: Int,
            permissions: Array<out String>,
            grantResults: IntArray
        ) {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults)
            if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) == PackageManager.PERMISSION_GRANTED
            ) {
                fetchImagesAndShowResult()
            } else {
                Snackbar.make(
                    findViewById(android.R.id.content),
                    "Can't get data without permission",
                    Snackbar.LENGTH_SHORT
                ).show()
            }
        }

        private fun fetchImagesAndShowResult() {
            val stringBuilder = StringBuilder()

            val projection = arrayOf( // media-database-columns-to-retrieve
                MediaStore.Images.ImageColumns.DISPLAY_NAME,
                MediaStore.Images.ImageColumns.DATE_MODIFIED
            )
        }
    }
}
```

```
val selection = null // sql-where-clause-with-placeholder-variables
val selectionArgs = null // values-of-placeholder-variables
val sortOrder =
    "${MediaStore.Images.ImageColumns.DATE_MODIFIED} DESC LIMIT 20" // sql-order-by-clause

applicationContext.contentResolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    projection,
    selection,
    selectionArgs,
    sortOrder
)?.use { cursor ->
    val nameColumn =
        cursor.getColumnIndexOrThrow(MediaStore.Images.ImageColumns.DISPLAY_NAME)
    val dateModifiedColumn =
        cursor.getColumnIndexOrThrow(MediaStore.Images.ImageColumns.DATE_MODIFIED)

    while (cursor.moveToNext()) {
        stringBuilder.append(cursor.getString(nameColumn)).append("\n")
            .append(dateFormat.format(cursor.getLong(dateModifiedColumn) * 1000L))
            .append("\n\n")
    }
    textResults.text = stringBuilder
}

companion object {
    const val TAG = "MainActivity"
    const val REQ_IMAGES = 0
}
}
```

При запуске приложения и нажатии кнопки загрузки данных, появиться запрос на доступ к фото и мультимедиа на устройстве:

MTS RUS



N 31 % 22:04

MediaStoreImageExample

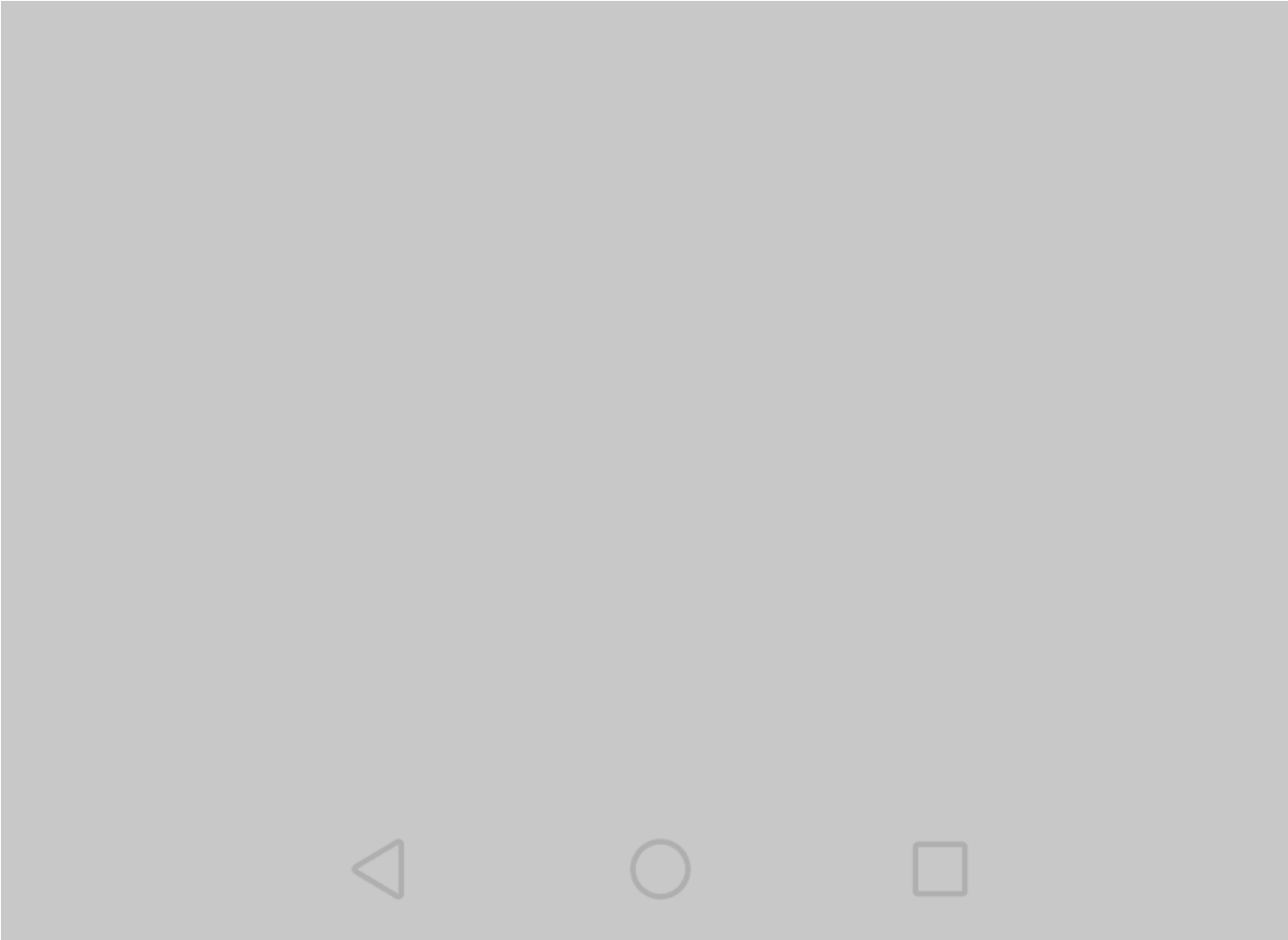
LOAD IMAGES






Открыть приложению
MediaStoreImageExample доступ к
фото и мультимедиа на устройстве?

РАЗРЕШИТЬ

ОТКЛОНИТЬ



После разрешения от пользователя загрузиться необходимая информация:

MTS RUS      31 %  22:04

MediaStoreImageExample

LOAD IMAGES

Screenshot_20210623_220435_com.google.android.permissioncontroller.jpg
23 июн. 2021 г. 22:04:35

Screenshot_20210623_220428_ru.samsung.itacademy.mdev.mediastoreimageexample.jpg
23 июн. 2021 г. 22:04:28

200295600510_113366.jpg
23 июн. 2021 г. 20:58:19

IMG-20210623-WA0019.jpg
23 июн. 2021 г. 20:06:41

IMG-20210623-WA0018.jpg
23 июн. 2021 г. 18:14:22

IMG-20210623-WA0015.jpg
23 июн. 2021 г. 14:31:31

IMG-20210623-WA0016.jpg
23 июн. 2021 г. 14:31:31

IMG-20210623-WA0017.jpg

23 июн. 2021 г. 14:31:31

IMG-20210623-WA0013.jpg

23 июн. 2021 г. 14:31:30

IMG-20210623-WA0014.jpg

23 июн. 2021 г. 14:31:30




IMG-20210623-WA0012.jpg

23 июн. 2021 г. 14:27:10

IMG-20210623-WA0010.jpg



Если мы изменим содержимое каталога с картинками, отображаемый результат также измениться:

MTS RUS      31 %  22:04

MediaStoreImageExample

LOAD IMAGES

Screenshot_20210623_220450_ru.samsung.itacademy
.mdev.mediastoreimageexample.jpg
23 июн. 2021 г. 22:04:50

Screenshot_20210623_220435_com.google.android.p
ermissioncontroller.jpg
23 июн. 2021 г. 22:04:35

Screenshot_20210623_220428_ru.samsung.itacademy
.mdev.mediastoreimageexample.jpg
23 июн. 2021 г. 22:04:28

200295600510_113366.jpg
23 июн. 2021 г. 20:58:19

IMG-20210623-WA0019.jpg
23 июн. 2021 г. 20:06:41

IMG-20210623-WA0018.jpg
23 июн. 2021 г. 18:14:22

IMG-20210623-WA0015.jpg
23 июн. 2021 г. 14:31:31

IMG-20210623-WA0016.jpg
23 июн. 2021 г. 14:31:31

IMG-20210623-WA0017.jpg
23 июн. 2021 г. 14:31:31

IMG-20210623-WA0013.jpg
23 июн. 2021 г. 14:31:30

IMG-20210623-WA0014.jpg
23 июн. 2021 г. 14:31:30



Полный код примера можно найти [здесь](#)

[Начать тур для пользователя на этой странице](#)