5.4. Обмен данными и файлами

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 5.4. Обмен данными и файлами

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 18:00

Оглавление

1. 5.4.1. Обмен простыми данными между приложениями

Одной из замечательных опций приложений для Android-приложений - это их способность общаться и интеграции друг с другом. Действительно, зачем изобретать функционал, который не является базисом для вашего приложения, когда он уже есть в другом?

Например, вы можете поделиться изображением из приложения Google Photo в приложение Facebook или Instagram, чтобы создать сообщение или историю. В настоящее время это обычная функция для приложенияй Android.

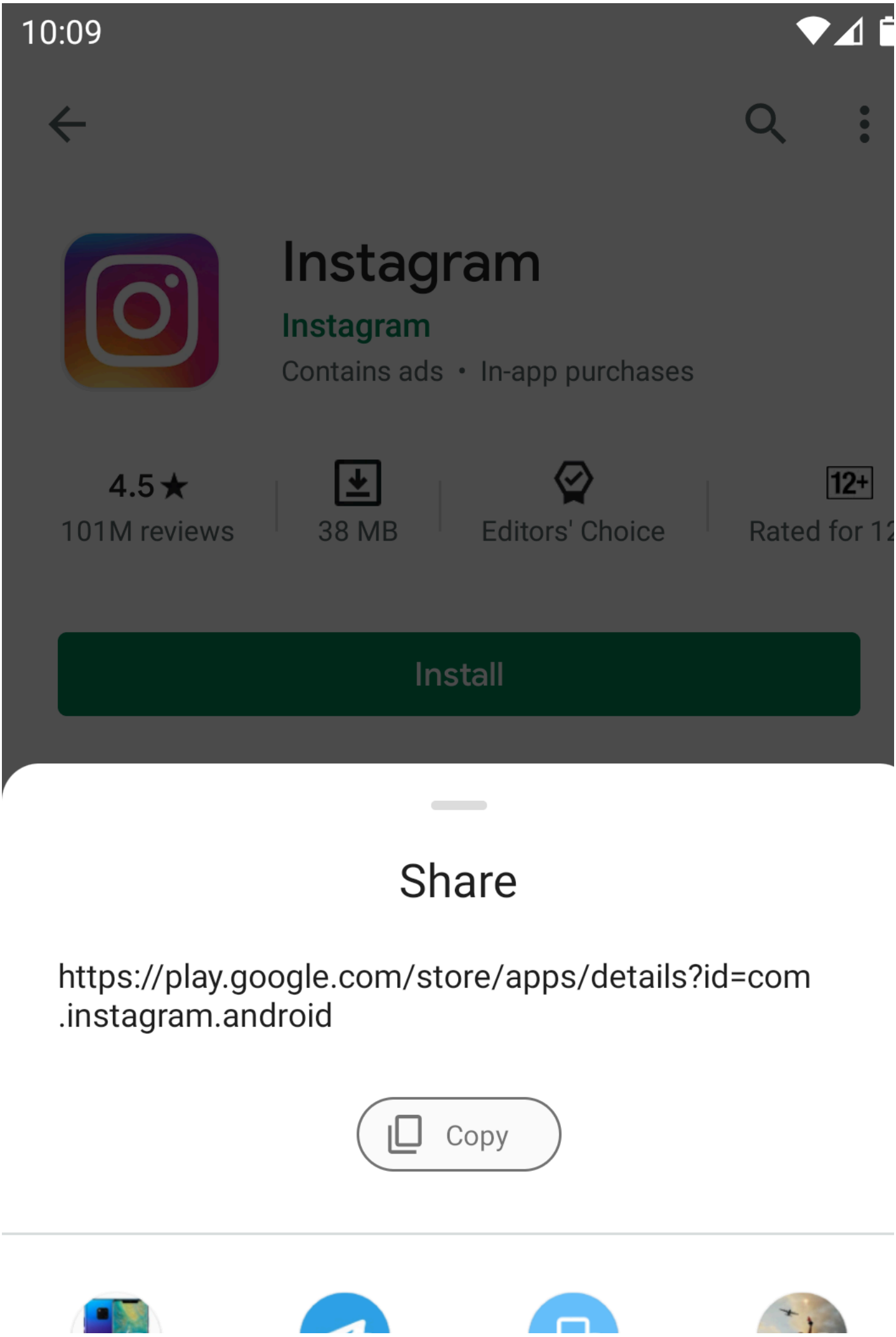
В данном уроке рассмотрим некоторые распространенные способы отправки и получения простых данных (например, изображений, текста или файлов) между приложениями с помощью таблицы общих ресурсов Android и намерений.

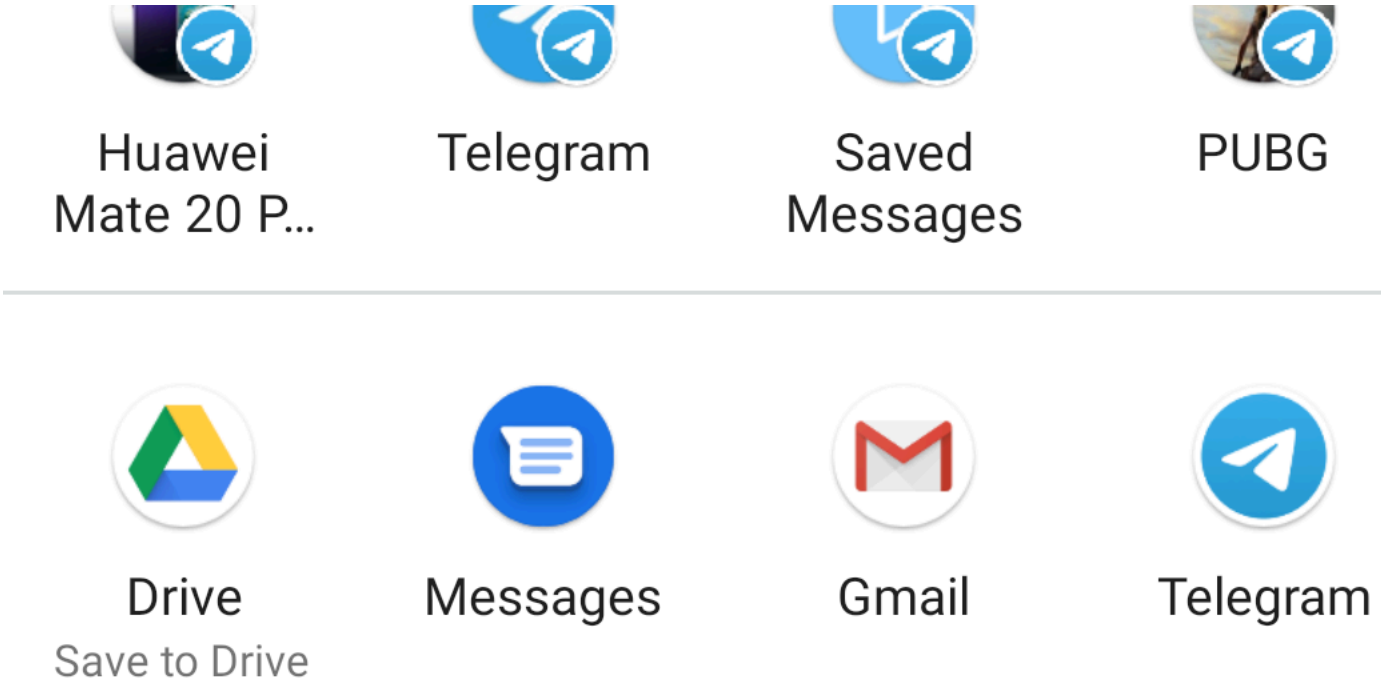
Отправка простых данных другим приложениям

Для отправки простых данных другим приложениям в Android используются намерения. Существует два способа обмена данными между приложениями:

- Общая таблица (Sharesheet) Android. Предназначена для отправки информации за пределы приложения и/или непосредственно другому пользователю. Например, совместное использование URL-адреса с другим пользователем.
- Распознаватель намерений (Intent Resolver) Android. В большей степени подходит для передачи данных на следующий этап четко определенной задачи.

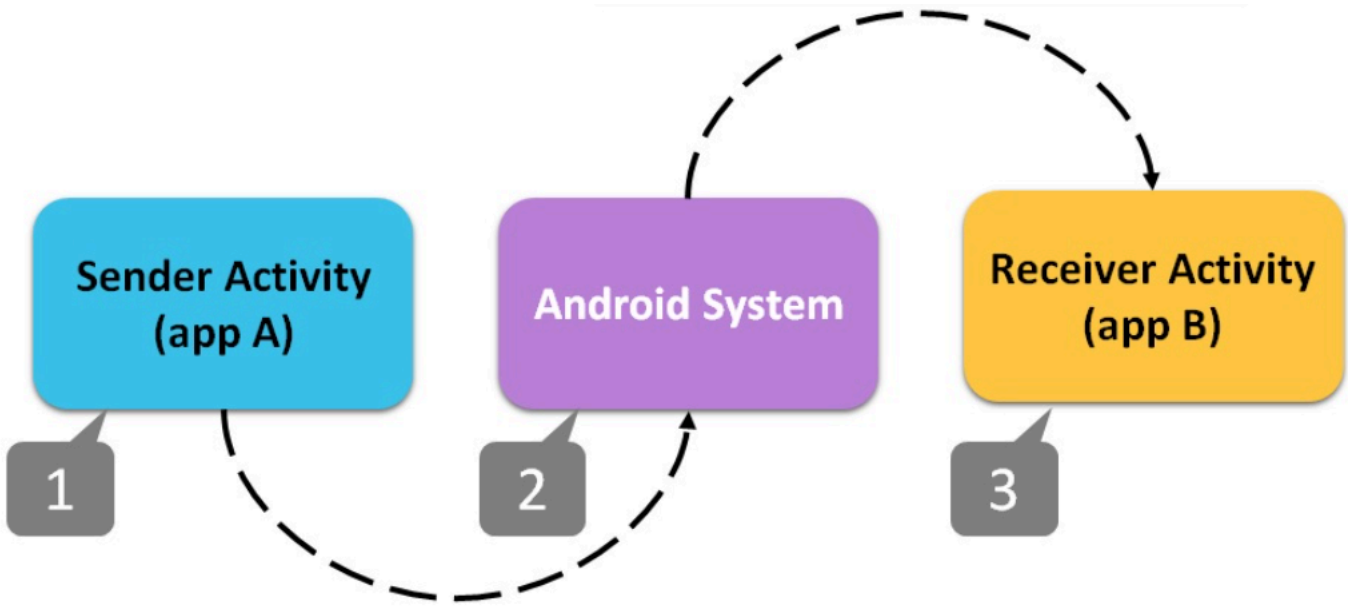
При создании намерения требуется указание действия, которое должно при этом выполнятся. Для отправки данных из одной активности в другую d в намерение необходимо передать константу [ACTION_SEND] (https://developer.android.com/reference/android/content/Intent#ACTION_SEND). При этом кому доставляются данные, не уточняется. Также необходимо указать данные и их тип. Система автоматически идентифицирует совместимые активности, которые могут получать данные, и отображает их для выбора пользователю. На экране приложения появляются доступные приложения для отправки контента.


10:09
Instagram
Instagram
Contains ads • In-app purchases
4.5★
101M reviews
38 MB
Editors' Choice
12+
Rated for 12
Install
Share
https://play.google.com/store/apps/details?id=com.instagram.android
Copy



В целях обеспечения согласованности действий пользователей в разных приложениях, при передачи данных разработчики рекомендуют все же использовать первый способ (Sharesheet). Такой метод передачи данных дает пользователям возможность делиться информацией с другими пользователями или приложениями всего в один клик.

Схематично обмен данными в приложении выглядит так, как показано на рисунке ниже:



Для передачи информации необходимо создать намерение и установить для его действия значение `Intent.ACTION_SEND`. Чтобы отобразить Sharesheet Android, вам нужно вызвать метод `Intent.createChooser()`.

Наиболее простое и распространенное использование Android Sharesheet - отправка текстового сообщения из одной активности в другую. Например, это может быть полезно полезно для обмена статьей или веб-сайтом с друзьями по электронной почте или в социальных сетях. Далее представлен фрагмент кода, где показано как это сделать:

```
val sendIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "Hello, world!!!")
    type = "text/plain"
}

val shareIntent = Intent.createChooser(sendIntent, null)
startActivity(shareIntent)
```

При необходимости в намерение можно добавить дополнительные сведения, например электронную почту получателя (`EXTRA_EMAIL`), тему письма (`EXTRA_SUBJECT`) и др.

С помощью действия `ACTION_SEND` есть возможность поделиться двоичными данными. Для этого необходимо установить соответствующий тип MIME и передать `Intent.EXTRA_STREAM` и URI ресурса в метод `putExtras()`. Обычно данный подход задействуют для совместного использования изображений, однако он может использоваться для совместного использования любого типа двоичного контента.

```
val shareIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_STREAM, uri)
    type = "image/jpeg"
}
startActivity(Intent.createChooser(shareIntent, resources.getText(R.string.send_to)))
```

Получающему приложению необходимо разрешение на доступ к данным, на которые указывает URI. Это можно сделать, используя один следующих способов:

- Хранение данных в своем собственном контент-провайдере (ContentProvider). При этом целесообразно убедиться, что другие приложения имеют правильное разрешение на доступ к нему. Разработчики предлагают в качестве предпочтительного механизма использование временных разрешений на URI и предоставление доступа только приложению-получателю. Простой способ такай ContentProvider- использовать класс FileProvider.
- Использование MediaStore Api. Хранилище MediaStore в основном используется для типов MIME видео, аудио и изображений. Но, начиная с Android 3.0 (уровень API 11), в нем можно хранить типы, не относящиеся к мультимедиа.

При отправке данных другому приложению вы должны указать наиболее конкретный тип MIME. Например, при совместном использовании обычного текста следует использовать text/plain. Тип MIME `*/*` использовать не рекомендуется. Далее приведем несколько распространенных типов MIME при отправке простых данных в Android:

- text/plain, text/rtf, text/html, text/json
- image/jpg, image/png, image/gif
- video/mp4, video/3gp
- application/pdf

Приложения принимающие сообщения должны их зарегистрировать как text/*, image/*, video/* и т.д. Более подробную информацию для типов MIME можно найти в реестре IANA.

Для того, чтобы отправить другому приложению несколько фрагментов с содержимым, необходимо передать в метод `putExtras()` аргументы `Intent.ACTION_SEND_MULTIPLE` вместе со списком уникальных идентификаторов ресурсов. Тип MIME при этом варьируется в зависимости от сочетания контента, которым вы делитесь. К примеру, если вы хотите сразу передать три изображения PNG, тип MIME также будет "image/png". При сочетании файлов изображения с различным расширением необходимо использовать "image/*". *Использование совместных MIME-типов возможно (используйте "*/*"), однако не рекомендовано поскольку получателю неясно, что должно быть отправлено. Анализ и обработка ваших данных зависит от принимающего приложения. Приведем пример:*

```
val uris: ArrayList<Uri> = arrayListOf(
        imageUri1,
        imageUri2
)

val shareIntent = Intent().apply {
    action = Intent.ACTION_SEND_MULTIPLE
    putParcelableArrayListExtra(Intent.EXTRA_STREAM, uris)
    type = "image/*"
}
startActivity(Intent.createChooser(shareIntent, "Share images to.."))
```

Начиная с Android 10 (уровень API 29), на Sharesheet Android отображается предварительный просмотр текста содержимого сообщения. Иногда данный текст труден для понимания. Для этого существует опция более детального просмотра содержимого присланного сообщения. Для того, чтобы принимающее приложение или пользователь понял, что за сообщение ему пришло, при предварительном просмотре текста можно задать его заголовок и миниатюру. Сделать это можно следующим образом:

```
val share = Intent.createChooser(Intent().apply {
      action = Intent.ACTION_SEND
      putExtra(Intent.EXTRA_TEXT, "hhttps://innovationcampus.ru/lms/mod/book/view.php?id=1132&amp;chapterid=1063")
      // заголовок содержимого
      putExtra(Intent.EXTRA_TITLE, "Introducing to data sharing!")
      // URI контента изображению для отображения
      data = contentUri
      flags = Intent.FLAG_GRANT_READ_URI_PERMISSION
  }, null)
  startActivity(share)
```

Более подробно об отправлении простых данных с использованием Sharesheet Android можно узнать в официальной документации.

Теперь перейдем ко второму способу отправки простых данных - распознаватель намерений (intent resolver). Его использование целесообразно при отправке данных в другое приложение в рамках определенного потока задач. Ключевым отличием от использования Sharesheet выступает отсутствие необходимости вызова метода `Intent.createChooser()`:

```
val sendIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "Hello!!!")
    type = "text/plain"
}
startActivity(sendIntent)
```

Прием простых данных другим приложениям

Приложение также может получать данные и из других приложений. При разработке приложений-приемников необходимо подумать, как пользователи взаимодействуют с вашим ними и какие типы данных необходимо получать от других приложений. Например, в мессенджере можно получать сообщения, фото, видео и т.д. Ранее было рассказано о двух способах оправления контента: через Sharesheet и intent resolver. Все полученные данные имеют тип MIME, заданный передающим приложением. Ваше приложение должно иметь возможность получать максимально широкий спектр типов MIME.

Для определения того, какие данные приложение готово принять существуют фильтры в манифесте. В приложениях-приемниках сообщений необходимо определить фильтр намерений в манифесте, используя элемент `<intent-filter>`. Например, если ваше приложение обрабатывает прием текста, одного или нескольких изображений любого типа, файл AndroidManifest.xml может выглядеть так:

```
activity android:name=".MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
</activity>
```

Когда приложение-отправитель попытается поделиться соответствующим контентом, ваше приложение будет в списке на предполагаемую рассылку. Если пользователь выберет ваше приложение для рассылки, будет запущена активность MyActivity из предыдущего примера.

Для получения необходимого намерения, в приложении-приемнике сообщения сначала необходимо вызвать метод `getIntent()`. Получив нужный объект и изучив его можно определить дальнейшие действия. При этом нужно помнить о необходимости обработки двоичных данных в отдельном потоке. Код в приложении-приемнике может быть следующим:

```
override fun onCreate(savedInstanceState: Bundle?) {
    //...
    when {
        intent?.action == Intent.ACTION_SEND -> {
            if ("text/plain" == intent.type) {
                handleSendText(intent) // обработка отправляемого текста
            } else if (intent.type?.startsWith("image/") == true) {
                handleSendImage(intent) // обработка одной отправляемой картинки
            }
        }
        intent?.action == Intent.ACTION_SEND_MULTIPLE
                && intent.type?.startsWith("image/") == true -> {
                handleSendMultipleImages(intent) // обработка множества отправляемых картинок
        }
        else -> {
            // обработка других намерений
        }
    }
    //...
}

private fun handleSendText(intent: Intent) {
    intent.getStringExtra(Intent.EXTRA_TEXT)?.let {
        // обновление UI для отображение отправленного текста
    }
}

private fun handleSendImage(intent: Intent) {
    (intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM) as? Uri)?.let {
        // обновление UI для отображение отправленного изображения
    }
}

private fun handleSendMultipleImages(intent: Intent) {
    intent.getParcelableArrayListExtra<Parcelable>(Intent.EXTRA_STREAM)?.let {
        // обновление UI для отображение множества отправленных изображений
    }
}
```

Обновление пользовательского интерфейса после получения данных зависит от функционала приложения.

2. 5.4.2. Обмен файлами между приложениями

Настройка общего доступа android приложений к файлам

Зачастую в приложениях необходимо отправлять файлы другому приложению. Например из галереи в редакторы изображений. Единственный безопасный способ отправить файл из приложения другому - это передать получающему приложению URI этого файла и предоставить временное разрешение на доступ нему.

В Android существует специальный класс FileProvider, в котором есть метод `getUriForFile()` для создания URI содержимого файла. `FileProvider` — это подкласс ContentProvider. Хотя `ContentProvider` — это компонент, который позволяет вам безопасно делиться любыми данными, `FileProvider` используется специально для совместного использования внутренних файлов приложения.

Чтобы FileProvider работал, нужно выполнить три следующих действия:

- Определить `FileProvider` в файле AndroidManifest.xml;
- Создать XML-файл, содержащий все пути, которые `FileProvider` будет использовать совместно с другими приложениями;
- Связать действительный URI в `Intent` и активировать его.

Чтобы определить `FileProvider` внутри AndroidManifest.xml, необходимо ознакомиться с этими атрибутами и элементами:

- `android:authorities`
- `android:exported`
- `android:grantUriPermissions`
- `android:name`
- `<meta-data>` субэлемент

Запись в манифесте указывает полномочия для использования при создании URI контента. В следующем фрагменте показано, как добавить в манифест элемент :

```
<manifest>
    ...
    <application>
        ...
        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="com.mydomain.fileprovider"
            android:exported="false"
            android:grantUriPermissions="true">
            ...
        </provider>
        ...
    </application>
</manifest>
```

После того, как вы определили `FileProvider` в своем *AndroidManifest.xml*, вы, наконец, готовы его использовать. Чтобы поделиться файлом, вам нужно создать намерение и предоставить ему действительный URI. URI генерируется с использованием класса `FileProvider`. В этом примере полномочием является поставщик файлов *com.mydomain.fileprovider*. Вы атрибуте `android:authorities` вы должны определить хотя бы одно уникальное полномочие. Система Android хранит список всех поставщиков, и она отличает их по полномочиям. Полномочие определяет `FileProvider` точно так же, как ID приложения определяет приложение для Android. В общем, Android-система использует определенную схему URI для `ContentProviders`. Схема определяется как *content:///*, поэтому система будет знать, какой `ContentProvider` запрашивается, сопоставляя полномочия URI с полномочиями `ContentProvider`.

Субэлемент `<meta-data>` указывает на XML-файл, в котором указаны каталоги, к которым вы хотите предоставить общий доступ. Атрибут `android:resource` - это путь и имя файла без расширения .xml.

После добавления провайдера в манифест приложения необходимо указать каталоги, содержащие файлы, которыми вы хотите поделиться. Чтобы указать каталоги, создайте файл с именем *filepaths.xml* в каталоге *res/xml/* проекта. В этом файле укажите каталоги, добавив элемент XML для каждого каталога. В следующем фрагменте показан пример содержимого *res/xml/filepaths.xml*.

```
<paths>
    <files-path path="images/" name="myimages" />
</paths>
```

XML-файл должен иметь элемент `<paths>` в качестве его корня. Элемент `<paths>` должен иметь как минимум один дочерний элемент, который может быть следующим:

- `<files-path/>` — внутреннее хранилище приложения,
- `<cache-path/>` — кэш внутреннего хранилища приложения,

- `<external-path/>` — публичное внешнее хранилище,
- `<external-files-path/>` — внешнее хранилище приложения,
- `<external-cache-path/>` — кэш внешнего хранилища приложения.

Как видно, они различаются в зависимости от каталога приложения, которое они определяют. Каждый элемент должен иметь атрибуты `path` и `name`. Атрибут `path`определяет подкаталог, который вы хотите использовать, и он не поддерживает подстановочные знаки. Атрибут name используется по соображениям безопасности, и он заменит ваше имя подкаталога на его значение.

Совместное использование файлов приложениями

После надлежащей настройки приложения для обмена файлами с использованием URI, появляется возможность отвечать на запросы других приложений для обмена файлами. Один из способов ответа - это серверное приложение (то есть ваше приложение). Интерфейс выбора файла с помощью серверного приложения смогут вызвать вызвать другие приложения. Для этого необходимо позволить запрашивающему приложению вызвать этот интерфейс для получения URI выбранного файла.

Ваше приложение должно обеспечить выбор файлов с использованием специальной Activity. Принимающие приложения запускают эту Activity с помощью вызова метода `startActivityForResult()` с `Intent` содержащим действие `ACTION_PICK`. Когда приложение-приемник вызывает метод `startActivityForResult()`, приложение-отправитель может вернуть результат клиентскому приложению в виде URI содержимого для файла, выбранного пользователем.

Настройку выбора файлов через Activity, следует начать с указания Activity в манифесте приложения. Также необходимо указать фильтр намерений, для действия `ACTION_PICK` и категориям `CATEGORY_DEFAULT` и `CATEGORY_OPENABLE`. Кроме того, следует добавить фильтр MIME-типа для файлов вашего приложения, которые будут использоваться в других приложениях. Таким образом конфигурация в манифесте может бвть следующей:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    ...
        <application>
        ...
            <activity
                android:name=".FileSelectActivity"
                android:label="@File Selector" >
                <intent-filter>
                    <action
                        android:name="android.intent.action.PICK"/>
                    <category
                        android:name="android.intent.category.DEFAULT"/>
                    <category
                        android:name="android.intent.category.OPENABLE"/>
                    <data android:mimeType="text/plain"/>
                    <data android:mimeType="image/*"/>
                </intent-filter>
            </activity>
```

Дальше можно определить класс `Activity`, который отображает файлы, доступные из каталога *files/images/* во внутреннем хранилище устройства. позвольте пользователю выбрать нужный файл. Далее в коде показано, как определить Activity и отвечать на выбор пользователя:

```
class MainActivity : Activity() {

    // Путь к корню внутреннего хранилища  приложения.
    private lateinit var privateRootDir: File
    // Путь к подкаталогу "images"
    private lateinit var imagesDir: File
    // Массив файлов в подкаталоге images
    private lateinit var imageFiles: Array<File>
    // Массив имен файлов, соответствующих imageFiles
    private lateinit var imageFilenames: Array<String>

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // Настройка намерения для отправки данных запрашивающим приложениям
        resultIntent = Intent("com.example.myapp.ACTION_RETURN_FILE")
        // Получение файлов / подкаталогов внутреннего хранилища
        privateRootDir = filesDir
        // Получение подкаталога files / images;
        imagesDir = File(privateRootDir, "images")
        // Получение файлов в подкаталоге images
        imageFiles = imagesDir.listFiles()
        // Set the Activity's result to null to begin with
        setResult(Activity.RESULT_CANCELED, null)
        // ...
    }
```

После того, как пользователь выберет файл, ваше приложение должно определить, какой это файл, и сгенерировать соответствующий URI. Так как Activity отображает список доступных файлов в ListView, когда пользователь щелкает имя файла система вызывает метод `onItemClick()`, в котором вы можете получить выбранный файл. Теперь в методе `onItemClick()`, необходимо получить объект File и передать его в в метод `getUriForFile()`, вместе

с полномочиями, указанными в элементе для FileProvider. В следующем фрагменте показано, как определить выбранный файл и получить для него URI содержимого:

```
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // Определите слушателя, который реагирует на щелчки по файлу в ListView
        fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
            // Предположим, что имена файлов находятся в массиве imageFilename.
            val requestFile = File(imageFilenames[position])

            // Используйте FileProvider, чтобы получить URI содержимого
            val fileUri: Uri? = try {
                FileProvider.getUriForFile(
                        this@MainActivity,
                        "com.example.myapp.fileprovider",
                        requestFile)
            } catch (e: IllegalArgumentException) {
                Log.e("File Selector",
                        "The selected file can't be shared: $requestFile")
                null
            }
            //...
        }
        //...
    }
```

Необходимо иметь ввиду, что вы можете сгенерировать только URI контента для файлов, находящихся в каталогах, указанных в файле мета-данных, содержащем -элементы. В случае вызова метода `getUriForFile()` для объекта File в каталоге, который не был указан, будет выброшен `IllegalArgumentException`.

Теперь, когда есть URI файла, доступ к которому необходимо предоставить другим приложениям, следует позволить принимающему приложению получить доступ к нему. Для этого, следует добавить URI в Intent, установив соответствующие флаги разрешений на Intent. Данные разрешения являются временными.

Следующий код демонстрирует, как предоставить доступ к файлу для чтения:

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    // Определите слушателя, который реагирует на щелчки по файлу в ListView
    fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
            //...
            if (fileUri != null) {
                // Предоставить временное разрешение на чтение URI контента
                resultIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
                //...
            }
            //...
        }
        //...
    }
```

Чтобы дать доступ к файлу приложению, которое его запросило, следует передать объект `Intent` содержащий URI файла и необходимые разрешения в метод `setResult()`. Для этого можно задействовать следующую логику в коде:

```
override fun onCreate(savedInstanceState: Bundle?) {
    //...
    // Определите слушателя, который реагирует на щелчки по файлу в ListView
    fileListView.onItemClickListener = AdapterView.OnItemClickListener { _, _, position, _ ->
        //...
        if (fileUri != null) {
            //...
            // Поместите тип Uri и MIME в Intent
            resultIntent.setDataAndType(fileUri, contentResolver.getType(fileUri))
            // Установите результат
             setResult(Activity.RESULT_OK, resultIntent)
         } else {
            resultIntent.setDataAndType(null, "")
            setResult(RESULT_CANCELED, resultIntent)
        }
    }
```

В случае когда приложению необходимо получить доступ к файлу, предоставляемому другим приложением, приложение отправляет запрос в приложение, предоставляющее файл. В большинстве случаев, запрос запускает Activity в приложении-отправителе, которая показывает файлы, которыми оно может поделиться. Пользователь выбирает файл, после чего сервер приложений возвращает URI контента для файла приложения-получателя.

Для запроса файла из отправителя, получателю необходимо вызвать метод `startActivityForResult` с намерением, содержащим действие, такое как `ACTION_PICK`, и тип MIME, который может обрабатывать приложение-получатель. Сделать это можно так, как показано в следующем фрагменте:

```
class MainActivity : Activity() {
    private lateinit var requestFileIntent: Intent
    private lateinit var inputPFD: ParcelFileDescriptor
    //...
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        requestFileIntent = Intent(Intent.ACTION_PICK).apply {
            type = "image/jpg"
        }
        //...
    }
   // ...
    private fun requestFile() {
       startActivityForResult(requestFileIntent, 0)
        //...
    }
   // ...
}
```

Приложение-отправитель посылает URI содержимого файла назад в приложение-получатель в намерении Intent. Это Intent передается в методе `onActivityResult()`. Как только получателю пришло URI содержимого файла, он может получить доступ к файлу, через `FileDescriptor`:

```
public override fun onActivityResult(requestCode: Int, resultCode: Int, returnIntent: Intent) {
    //Если выбор не сработал
    if (resultCode != Activity.RESULT_OK) {
        // Выйти, ничего не делая
        return
    }
    // Получить URI содержимого файла из входящего Intent
    returnIntent.data?.also { returnUri ->
        /*
         * Попытка открыть файл для чтения, используя возвращенный URI.
         */
        inputPFD = try {
            contentResolver.openFileDescriptor(returnUri, "r")
        } catch (e: FileNotFoundException) {
            e.printStackTrace()
            Log.e("MainActivity", "File not found.")
            return
        }

        // обычный файловый дескриптор для файла
        val fd = inputPFD.fileDescriptor
        ...
    }
}
```

При этом метод `openFileDescriptor()` возвращает объект `ParcelFileDescriptor`. Из этого объекта, приложение-получатель извлекает объект `FileDescriptor`, который оно может использовать для чтения необходимого файла.

Получение информации о файле

Прежде чем приложение-получатель будет работать с файлом, для которого у него есть URI, приложение может запросить информацию о файле у отправителя (например, тип данных и размер файла). Тип данных помогает получателю определить, может ли оно обрабатывать файл, а размер файла помогает ему настроить буферизацию и кэширование файла.

Тип данных файла показывает как приложению-получателю обрабатывать содержимое файла. Для поучения типа данных файла по его URI, получателю следует вызывать метод `ContentResolver.getType()`, который возвращает MIME-тип файла. По умолчанию `FileProvider` определяет MIME файла по его расширению. В следующем фрагменте кода показано как извлечь MIME-тип файла:

```
val mimeType: String? = returnIntent.data?.let { returnUri ->
        contentResolver.getType(returnUri)
}
```

По умолчанию в классе `FileProvider` есть реализация для метода `query()`, который возвращает имя и размер файла, связанного с URI содержимого в объекте Cursor. Реализация по умолчанию возвращает два столбца:

- `DISPLAY_NAME` - имя файла (String). Значение такое же, как значение, возвращаемое метом `File.getName()`.
- `SIZE` - размер файла в байтах (long) Это значение такое же, как значение, возвращаемое методом `File.length()`.

Приложение-получатель файла может получить DISPLAY_NAME и SIZE, установив для всех аргументов метода `query()` значение null, за исключением URI. Например, этот фрагмент кода извлекает `DISPLAY_NAME` и `SIZE` файла и отображает их в TextView:

```
returnIntent.data?.let { returnUri ->
   contentResolver.query(returnUri, null, null, null, null)}?.use { cursor ->
        val nameIndex = cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME)
        val sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE)
        cursor.moveToFirst()
        findViewById<TextView>(R.id.filename_text).text = cursor.getString(nameIndex)
        findViewById<TextView>(R.id.filesize_text).text = cursor.getLong(sizeIndex).toString()
 }
```

3. 5.4.4. Упражнение 5.4.

Разработаем простое приложение, которое может делиться содержимым, таким как URL, текст и файлы изображений с другими приложениями, установленными на вашем устройстве Android. Например, вы можете поделиться изображением из приложения Google Photo в приложении Facebook или Instagram, чтобы создать сообщение или историю.

Подготовьте XML-макет, activity_main.xml будет состоять из двух кнопок для отправки текстового сообщения и файла изображения:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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/buttonShareTextUrl"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:text="@string/share_text_or_url" />

    <Button
        android:id="@+id/buttonShareImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/buttonShareTextUrl"
        android:text="@string/share_image" />
</RelativeLayout>
```

Экран приложения будет выглядеть следующим образом:

Внутри метода onCreate файла MainActivity.kt поместите кнопки и обработчики `OnClickListener`.

```
val handler: View.OnClickListener = View.OnClickListener { v ->
    when (v.id) {
        R.id.buttonShareTextUrl -> shareTextUrl()
        R.id.buttonShareImage -> shareImage()
    }
}

findViewById<View>(R.id.buttonShareTextUrl).setOnClickListener(handler)
findViewById<View>(R.id.buttonShareImage).setOnClickListener(handler)
```

Теперь определим метод `shareTextUrl()`, который будет запускаться каждый раз, когда пользователь нажимает кнопку «Share Text or URL». В методе данные добавляются в намерение, а приложение-получатель решит, что с ними делать.

```
 private fun shareTextUrl() {
        val share = Intent(Intent.ACTION_SEND)
        share.type = "text/plain"
        share.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)

        share.putExtra(Intent.EXTRA_SUBJECT, "Hello from my app")
        share.putExtra(Intent.EXTRA_TEXT, "http://myitschool.ru")
        startActivity(Intent.createChooser(share, "Share link!"))
        Toast.makeText(applicationContext, "Text shared", Toast.LENGTH_LONG).show()
    }
```

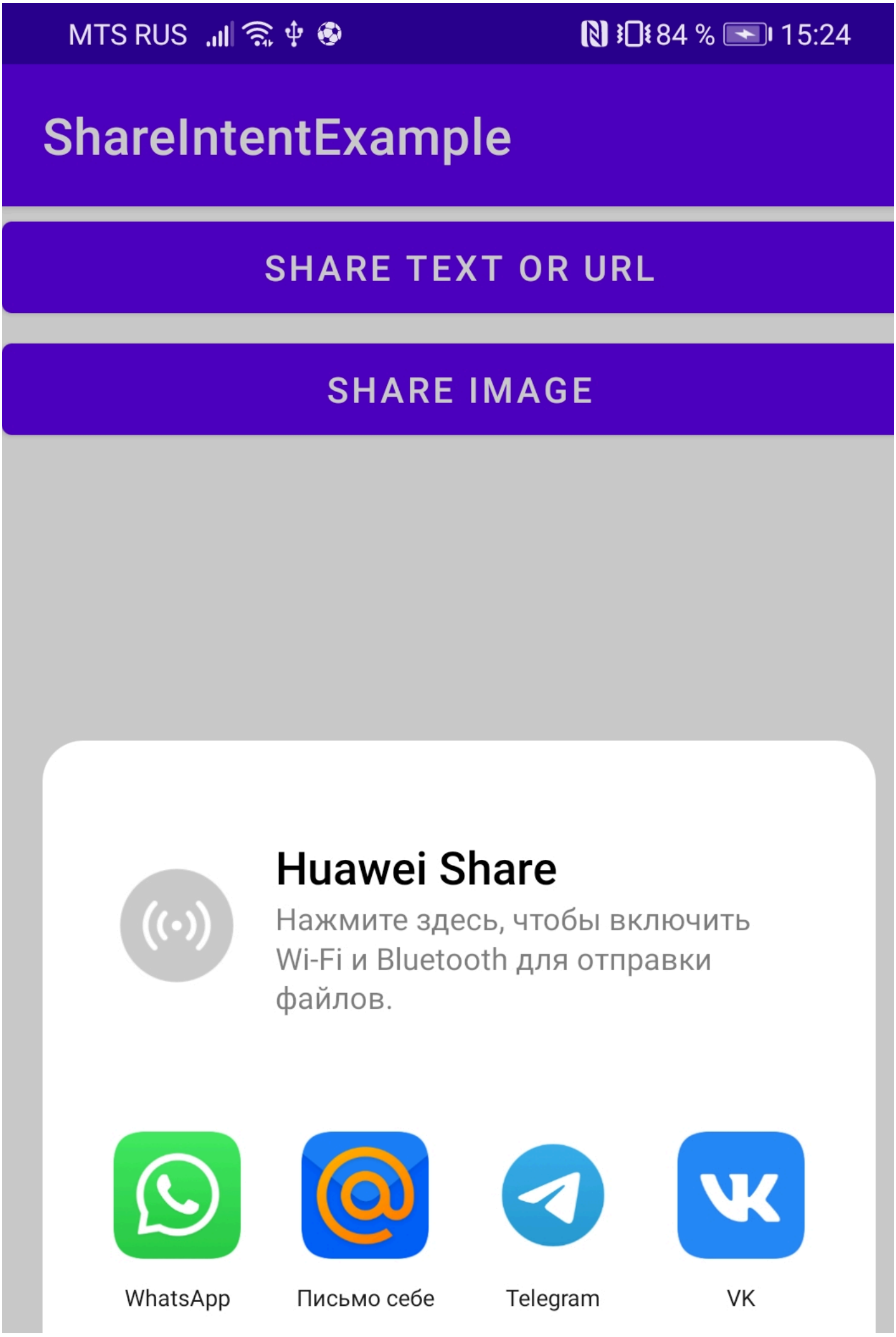
Также необходимо реализовать метод `shareImage()` для обмена изображениями. Он запускается, когда пользователь нажимает кнопку «Share Image».

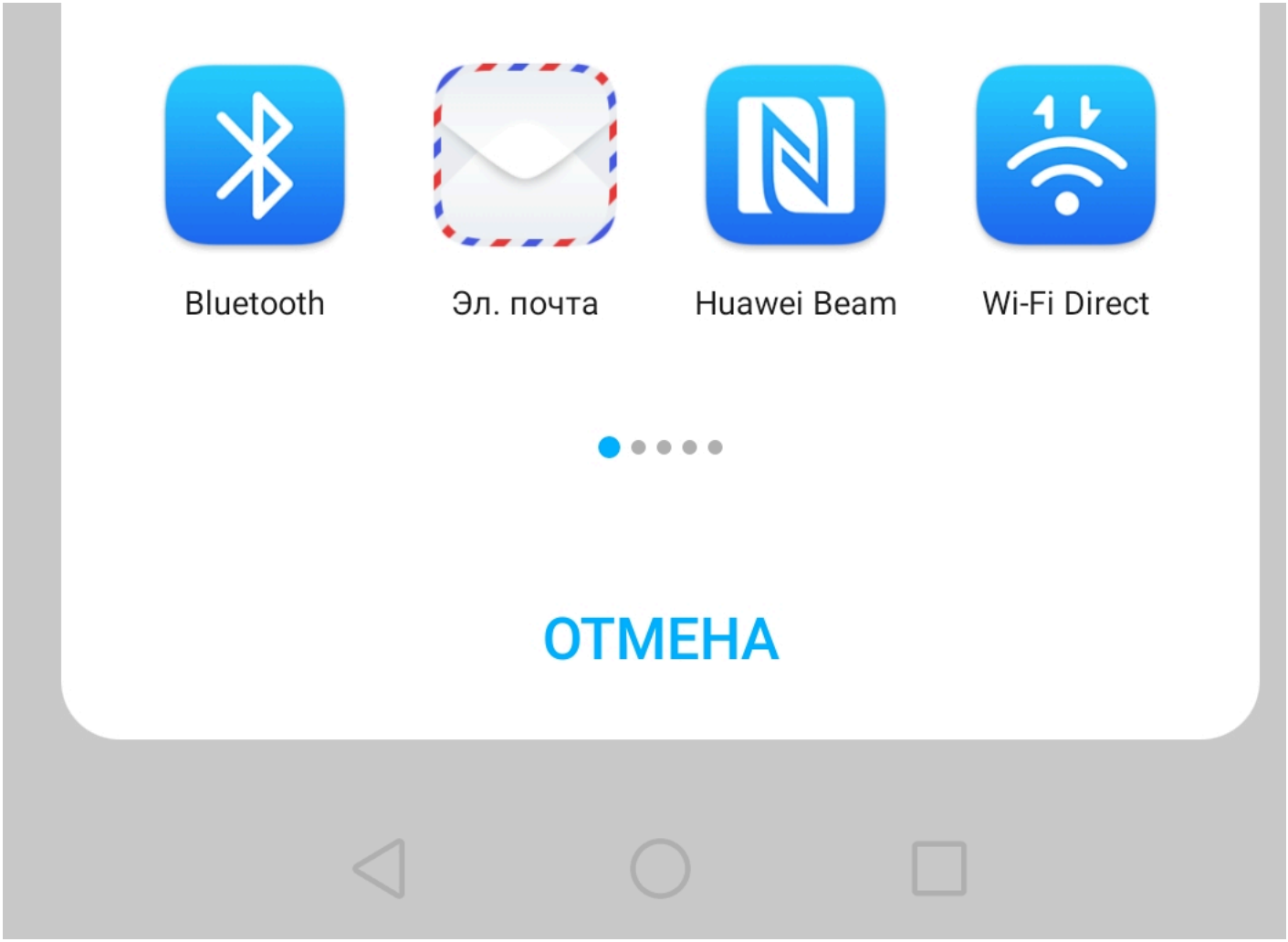
```
private fun shareImage() {
        val share = Intent(Intent.ACTION_SEND)

        share.type = "image/*"

        // Make sure you put example png image named myImage.png in your directory
        val imagePath: String = getExternalFilesDir(Environment.DIRECTORY_PICTURES)
                .toString() + "/myImage.png"
        val imageFileToShare = File(imagePath)
        val uri: Uri = Uri.fromFile(imageFileToShare)
        share.putExtra(Intent.EXTRA_STREAM, uri)
        startActivity(Intent.createChooser(share, "Share Image!"))
        Toast.makeText(applicationContext, "Image shared", Toast.LENGTH_LONG).show()
    }
```

Если вы хотите поделиться только изображением png, вы можете написать следующее: `setType("image / png")`, а для jpeg: `setType ("изображение / jpeg")`. Для корректной работы приложения убедитесь, что вы поместили изображение png с именем myImage.png в свой каталог. В качестве самостоятельной работы можно переделать код приложения так, чтобы он отправлял выбранный вами файл (причем не только файл изображения). Полностью код MainActivity будет следующим:

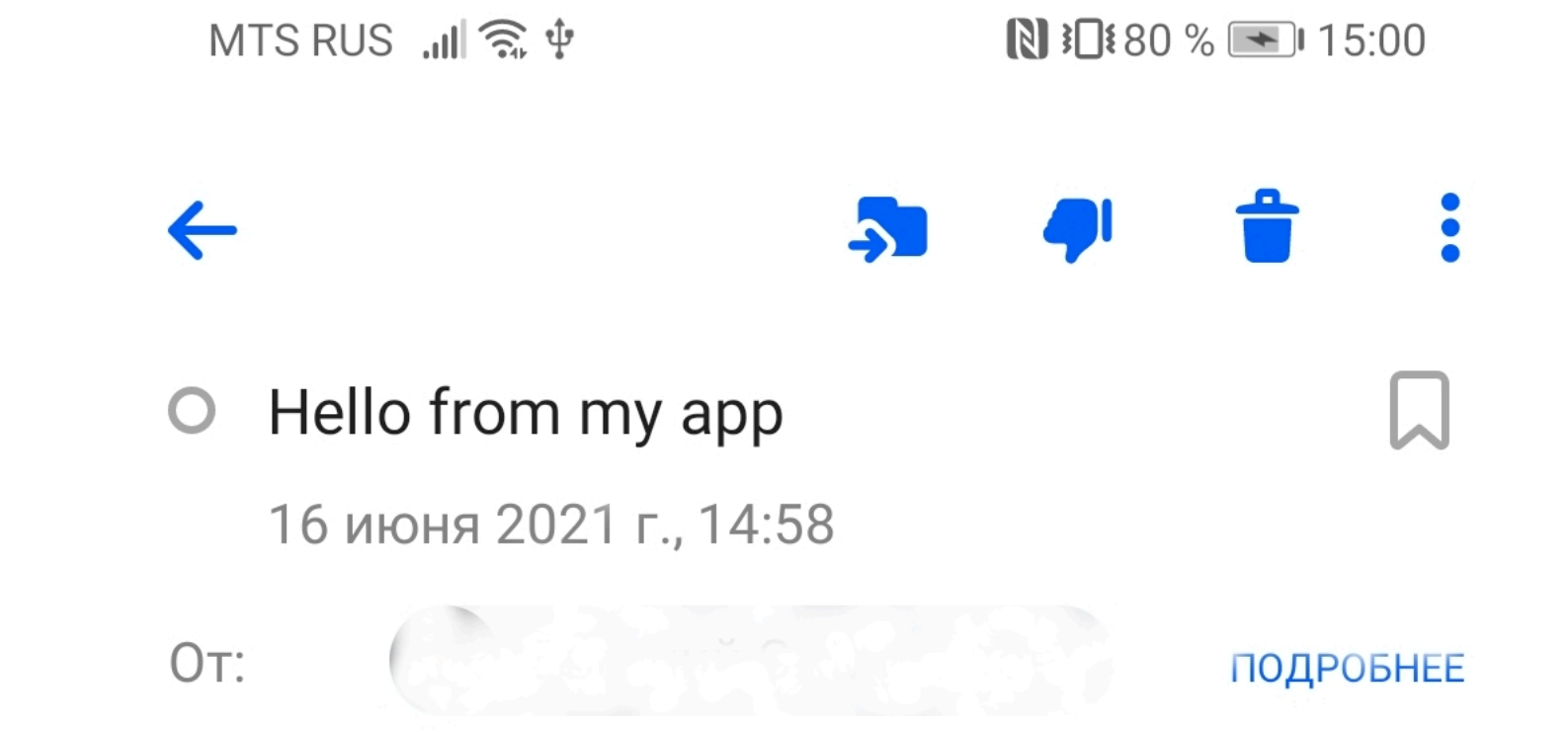
```
package ru.samsung.itacademy.mdev.shareintentexample

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.os.Environment
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import java.io.File

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val handler: View.OnClickListener = View.OnClickListener { v ->
            when (v.id) {
                R.id.buttonShareTextUrl -> shareTextUrl()
                R.id.buttonShareImage -> shareImage()
            }
        }

        findViewById<View>(R.id.buttonShareTextUrl).setOnClickListener(handler)
        findViewById<View>(R.id.buttonShareImage).setOnClickListener(handler)
    }

    private fun shareTextUrl() {
        val share = Intent(Intent.ACTION_SEND)
        share.type = "text/plain"
        share.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)

        share.putExtra(Intent.EXTRA_SUBJECT, "Hello from my app")
        share.putExtra(Intent.EXTRA_TEXT, "http://myitschool.ru")
        startActivity(Intent.createChooser(share, "Share link!"))
        Toast.makeText(applicationContext, "Text shared", Toast.LENGTH_LONG).show()
    }

    private fun shareImage() {
        val share = Intent(Intent.ACTION_SEND)

        share.type = "image/*"

        val imagePath: String = getExternalFilesDir(Environment.DIRECTORY_PICTURES)
                .toString() + "/myImage.png"
        val imageFileToShare = File(imagePath)
        val uri: Uri = Uri.fromFile(imageFileToShare)
        share.putExtra(Intent.EXTRA_STREAM, uri)
        startActivity(Intent.createChooser(share, "Share Image!"))
        Toast.makeText(applicationContext, "Image shared", Toast.LENGTH_LONG).show()
    }
}
```

При запуске приложения и нажатии одной из кнопок, откроется список других приложений, куда можно отправить соответствующую информацию.


MTS RUS
84 %
15:24
ShareIntentExample
SHARE TEXT OR URL
SHARE IMAGE
Huawei Share
Нажмите здесь, чтобы включить Wi-Fi и Bluetooth для отправки файлов.
WhatsApp
Письмо себе
Telegram
VK



При выборе, например, пункта "Письмо себе", на почту должно прийти письмо со следующим содержимым:



Hello from my app

16 июня 2021 г., 14:58

От:

ПОДРОБНЕЕ

http://myitschool.ru

--

Полный код приложения в упражнении можно найти по ссылке

Начать тур для пользователя на этой странице