Давайте научим наше приложение добавлять изображения к заметке. Как это будет происходить?

При редактировании заметки пользователь может нажать на кнопку прикрепления изображения, и получит выбор:

1. Выбрать изображение из галереи.
2. Сделать снимок с камеры.

После того, как мы получили от пользователя изображение, копируем его во внутреннее хранилище файлов и создаём запись в таблице SQLite (которую мы создадим позже) с информацией об этом изображении.

При удалении изображения удаляем запись из таблицы и файл из хранилища. Приступим к реализации!

Диалог выбора изображения

Добавьте в меню CreateNoteActivity элемент, при нажатии на который будет показан диалог выбора изображения.

Сделайте это самостоятельно по примеру тех элементов, которые мы добавляли ранее. В качестве иконки я выбрал "скрепку" из стандартного набора, но вы вольны выбрать ту иконку, которую считаете нужной.

Добавьте метод в код CreateNoteActivity такой метод:

private void showImageSelectionDialog() {

}

И добавим вызов метода в onOptionsItemSelected():

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

// ...

case R.id.action\_attach:

showImageSelectionDialog();

return true;

default:

return super.onOptionsItemSelected(item);

}

В этом методе мы будем показывать диалог с вариантами получения изображения: из галереи или с камеры.

Для отображения диалогов в Android обычно используют класс AlertDialog. Один из вариантов использования диалога — отображение списка. В случае со списком нам нужно заранее задать этот список в ресурсах, например, в strings.xml:

<string-array name="attachment\_variants">

<item>Выбрать из галереи</item>

<item>Сделать снимок</item>

</string-array>

Теперь в методе showImageSelectionDialog() создадим и покажем диалог:

private void showImageSelectionDialog() {

AlertDialog alertDialog = new AlertDialog.Builder(this)

.setTitle(R.string.title\_dialog\_attachment\_variants)

.setItems(R.array.attachment\_variants, new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

}

})

.create();

if (!isFinishing()) {

alertDialog.show();

}

}

Всё достаточно просто — создаём билдер диалога, в качестве единственного параметра конструктора передаём контекст.

Далее устанавливаем заголовок диалога (я заранее задал в ресурсах текст заголовка — сделайте это самостоятельно).

Потом устанавливаем список элементов и **Listener**, который будет использоваться при клике на элемент списка (второй параметр onClick() — это индекс элемента, на который кликнул пользователь).

Ну и создаём диалог, вызвав create(), после чего показываем его.

Перед показом важно убедиться, что Activity не завершается, иначе приложение крашнется — мы проверяем это вызовом метода isFinishing().

Создайте два новых метода:

private void pickImageFromGallery() {

}

lang=java

private void takePhoto() {

}

В методе onClick() в диалоге смотрим, что выбрал пользователь и вызываем соответствующий метод:

@Override

public void onClick(DialogInterface dialog, int which) {

if (which == 0) {

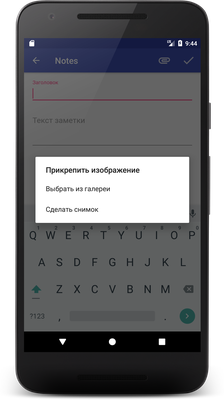
pickImageFromGallery();

} else if (which == 1) {

takePhoto();

}

}

[](https://skillberg.com/media/uploads/2017/11/17/d94e31c698874a7698b865b6f8b5ca0c.png)AlertDialog со списком вариантов

Выбор изображения из галереи

Выбор из галереи происходит уже знакомым нам образом:

1. Кидаем Intent, в котором сообщаем, что хотим получить от какого-то приложения изображение. Пользователь самостоятельно выбирает, какое приложение использовать.
2. Получаем результат в onActivityResult().

Как вы помните, при запуске Activity для получения какого-то результата нам потребуется некая константа, с помощью которой мы сможем понять в onActivityResult() откуда именно к нам пришёл результат. Зададим эту константу:

private static final int REQUEST\_CODE\_PICK\_FROM\_GALLERY = 1;

Теперь в методе pickImageFromGallery() начнём процедуру выбора изображения:

private void pickImageFromGallery() {

Intent intent = new Intent(Intent.ACTION\_PICK);

intent.setType("image/\*");

startActivityForResult(intent, REQUEST\_CODE\_PICK\_FROM\_GALLERY);

}

Переопределим метод onActivityResult():

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == REQUEST\_CODE\_PICK\_FROM\_GALLERY

&& resultCode == RESULT\_OK

&& data != null) {

// Получаем URI изображения

Uri imageUri = data.getData();

if (imageUri != null) {

try {

// Получаем InputStream, из которого будем декодировать Bitmap

InputStream inputStream = getContentResolver().openInputStream(imageUri);

// Декодируем Bitmap

final Bitmap bitmap = BitmapFactory.decodeStream(inputStream);

Log.i("Test", "Bitmap size: " + bitmap.getWidth() + "x" + bitmap.getHeight());

} catch (FileNotFoundException e) {

e.printStackTrace();

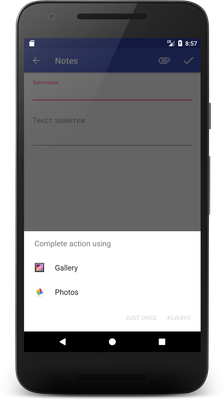
}

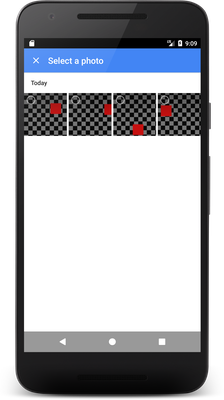
}

}

}

Проверяем, правильно ли всё работает:

[](https://skillberg.com/media/uploads/2017/11/17/e16ed77556de416c9cbc9ab870d9eb9b.png)Выбор приложения для получения изображения

[](https://skillberg.com/media/uploads/2017/11/17/626c89bdd95e4e3eaa4ffb04bbd9a6e6.png)Выбор изображения с использованием приложения Photos

Выбрав изображение, видим в логах:

11-05 09:09:52.223 5509-5509/com.skillberg.notes I/Test: Bitmap size: 640x480

Работает!

Получаем изображение с камеры

Теперь пришло время научить приложение получать изображение с камеры.

Хотя этот процесс и похож на получение изображения из галереи, его реализация потребует некоторых дополнительных усилий.

Выглядит он так:

1. Создаём файл, в который камера сохранит изображение.
2. Кидаем Intent, в котором передаём этот файл.
3. Получаем коллбэк в onActivityResult().

Получается, добавился ещё один шаг — нужно создать файл и передать его камере.

У каждого приложения есть возможность хранить свои файлы на SD-карте. Эти файлы могут быть как общедоступными, так и приватными. Пожалуй, мы выберем вариант с приватными файлами.

Получить доступ к директории можно используя метод Context.getExternalFilesDir(). Этот метод принимает в качестве параметра тип директории, которую мы хотим получить. Варианты таковы:

* DIRECTORY\_MUSIC
* DIRECTORY\_PODCASTS
* DIRECTORY\_RINGTONES
* DIRECTORY\_ALARMS
* DIRECTORY\_NOTIFICATIONS
* DIRECTORY\_PICTURES
* DIRECTORY\_MOVIES
* DIRECTORY\_DOWNLOADS
* DIRECTORY\_DCIM
* DIRECTORY\_DOCUMENTS

Нам вполне подойдёт вариант DIRECTORY\_PICTURES.

Создаём файл:

@Nullable

private File createImageFile() {

// Генерируем имя файла

String filename = System.currentTimeMillis() + ".jpg";

// Получаем приватную директорию на карте памяти для хранения изображений

// Выглядит она примерно так: /sdcard/Android/data/com.skillberg.notes/files/Pictures

// Директория будет создана автоматически, если ещё не существует

File storageDir = getExternalFilesDir(Environment.DIRECTORY\_PICTURES);

// Создаём файл

File image = new File(storageDir, filename);

try {

if (image.createNewFile()) {

return image;

}

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

На Android версий ниже, чем **4.4** для доступа к приватным файлам потребуется разрешение WRITE\_EXTERNAL\_STORAGE. Начиная с 4.4 это разрешение больше не требуется, поскольку эти файлы доступны только приложению, создавшему их.

Добавим разрешение в манифест:

<uses-permission

android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"

android:maxSdkVersion="18" />

Атрибут android:maxSdkVersion="18" говорит о том, что мы запрашиваем разрешение только для версий Android ниже **4.4**.

Ещё один важный момент, связанный с файлами: мы будем передавать камере ссылку на файл в виде URI, а, как вы помните из прошлых уроков, начиная с Android 7 для этого требуется FileProvider.

Создадим FileProvider в манифесте:

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="com.skillberg.notes.fileprovider"

android:exported="false"

android:grantUriPermissions="true">

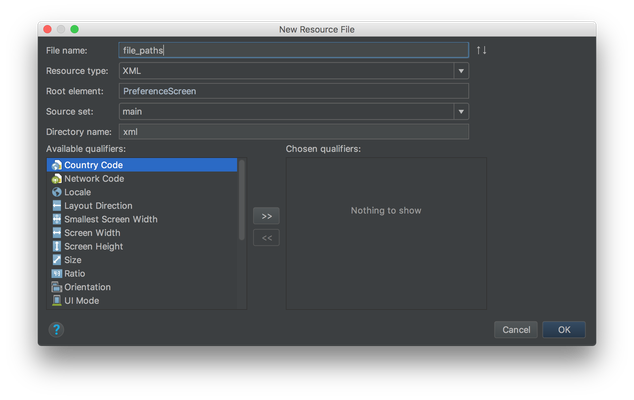
<meta-data

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths" />

</provider>

Для того, чтобы оно сработало, потребуется создать в ресурсах файл file\_paths.xml:

[](https://skillberg.com/media/uploads/2017/11/17/ab3ed1a6000e44daa7de82fc4a770021.png)Создание ресурса file\_paths.xml

<?xml version="1.0" encoding="utf-8"?>

<paths>

<external-path

name="my\_images"

path="Android/data/com.skillberg.notes/files/Pictures" />

</paths>

Android/data/com.skillberg.notes/files/Pictures — это относительный путь директории на SD-карте, к которой мы предоставляем доступ камере. Полностью он выглядит, например, так: /storage/emulated/0/Android/data/com.skillberg.notes/files/Pictures.

Теперь перейдём, непосредственно, к запуску камеры. Как и в случае с выбором изображения из галереи, нам понадобится константа с кодом запроса для камеры:

private static final int REQUEST\_CODE\_TAKE\_PHOTO = 2;

Обратите внимание на то, что константы должны иметь разное значение!

Так пригодится поле для файла, в который камера сохранит изображение, поскольку мы не получим этой информации из коллбэка:

private File currentImageFile;

Далее всё просто:

private void takePhoto() {

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

// Создаём файл для изображения

currentImageFile = createImageFile();

if (currentImageFile != null) {

// Если файл создался — получаем его URI

Uri imageUri = FileProvider.getUriForFile(this,

"com.skillberg.notes.fileprovider",

currentImageFile);

// Передаём URI в камеру

intent.putExtra(MediaStore.EXTRA\_OUTPUT, imageUri);

startActivityForResult(intent, REQUEST\_CODE\_TAKE\_PHOTO);

}

}

Осталось последнее: декодировать Bitmap из файла:

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == REQUEST\_CODE\_PICK\_FROM\_GALLERY

&& resultCode == RESULT\_OK

&& data != null) {

// ...

} else if (requestCode == REQUEST\_CODE\_TAKE\_PHOTO

&& resultCode == RESULT\_OK) {

Bitmap bitmap = BitmapFactory.decodeFile(currentImageFile.getAbsolutePath());

Log.i("Test", "Bitmap size: " + bitmap.getWidth() + "x" + bitmap.getHeight());

}

}

Готово!

Обновляем базу данных

Получать изображения мы научились, но нужно ещё их и связывать с заметками. Для этого мы создадим ещё одну таблицу, в которой будем хранить данные об этих связях:

* id — идентификатор изображения.
* path — путь до изображения.
* note\_id — идентификатор заметки, foreign key.

В NotesContract создайте внутренний класс по примеру класса Notes:

public static abstract class Images implements BaseColumns {

public static final String TABLE\_NAME = "images";

public static final Uri URI = Uri.parse(NotesContract.URI + "/" + TABLE\_NAME);

public static final String COLUMN\_PATH = "path";

public static final String COLUMN\_NOTE\_ID = "note\_id";

public static final String CREATE\_TABLE = String.format("CREATE TABLE %s " +

"(%s INTEGER PRIMARY KEY, " +

"%s TEXT NOT NULL, " +

"%s INTEGER NOT NULL, " +

"FOREIGN KEY (%s) REFERENCES %s (%s) ON DELETE CASCADE);",

TABLE\_NAME,

\_ID,

COLUMN\_PATH,

COLUMN\_NOTE\_ID,

COLUMN\_NOTE\_ID,

Notes.TABLE\_NAME,

Notes.\_ID);

public static final String[] PROJECTION = {

\_ID,

COLUMN\_PATH,

COLUMN\_NOTE\_ID

};

/\*\*

\* Типы данных

\*/

public static final String URI\_TYPE\_IMAGE\_DIR = "vnd.android.cursor.dir/vnd.skillberg.image";

public static final String URI\_TYPE\_IMAGE\_ITEM = "vnd.android.cursor.item/vnd.skillberg.image"

}

Этот класс принципиально не отличается от Notes ничем, кроме запроса создания таблицы — в нём мы создаём Foreign Key. Запрос будет выглядеть вот так:

CREATE TABLE images

(\_id INTEGER PRIMARY KEY,

path TEXT NOT NULL,

note\_id INTEGER NOT NULL,

FOREIGN KEY (note\_id) REFERENCES notes (\_id) ON DELETE CASCADE);

ON DELETE CASCADE говорит о том, что если заметка, связанная с изображением будет удалена, то изображение так же будет автоматически удалено.

Массив NotesContract.CREATE\_DATABASE\_QUERIES нужно дополнить запросом для создания таблицы с изображениями:

public static final String[] CREATE\_DATABASE\_QUERIES = {

Notes.CREATE\_TABLE,

Notes.CREATE\_UPDATED\_TS\_INDEX,

Images.CREATE\_TABLE

};

Константа DB\_VERSION также изменилась — теперь она равна 2, так как мы изменили структуру БД:

public static final int DB\_VERSION = 2;

Предположим, что мы уже реализовали создание и удаление изображений в БД. Работать оно, однако, не будет — приложение будет попросту крашиться.

Несмотря на то, что мы вроде как задали структуру таблиц, структура существующей БД ещё не обновилась. Есть два варианта обновления:

1. Переустановить приложение. Такой вариант подойдёт для разработчика, но не для пользователя — ведь все данные потеряются.
2. Реализовать обновление БД — вам обязательно нужно научиться делать это.

В классе NotesDbHelper есть метод onUpgrade(), который пока что пустует. В него передаются три параметра:

* sqLiteDatabase — база данных, которую обновляем.
* oldVersion — текущая версия БД.
* newVersion — версия БД, до которой обновляем.

БД ориентируется на версию, которую мы передаём в конструкторе, и если она меняется — вызывается этот метод. В нём мы должны ориентируясь на версии выполнить запросы, которые изменят структуру таблиц нужным нам образом.

Мы обновляемся с версии 1 до 2, и нам нужно выполнить запрос создания таблицы images. Делается это просто:

@Override

public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {

if (oldVersion == 1 && newVersion == 2) {

sqLiteDatabase.execSQL(NotesContract.Images.CREATE\_TABLE);

}

}

Вот и всё! В следующем уроке мы модифицируем ContentProvider для работы с изображениями.

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/NotesAndroid/tree/lesson-26)