Итак, мы продолжаем создание приложения, которое позволяет просматривать список установленных приложений, а так же удалять их и устанавливать новые. Мы остановились на том, что начали реализовывать Activity, которая позволит нам выбрать файл для установки.

Прежде чем предоставить пользователю список файлов для выбора, нужно получить этот список.

Система разрешений Android

Приложение для Android может получить доступ к множеству вещей в системе, одна из этих вещей — файлы.

Для того, чтобы пользователь знал, что именно может делать приложение, а так же мог контролировать доступ приложения к функциям системы существует система разрешений.

Приложение должно запросить разрешение (Permission) пользователя, прежде чем оно получит доступ к определенным функциям системы. Разрешения могут быть явными или неявными, метод получения зависит от типа разрешения. Существует два типа разрешений:

* **Обычные** — разрешения, требующиеся для получения данных, которые не опасны для "чувствительных" данных пользователя. Например, разрешение для доступа в интернет. Такие разрешения просто декларируются в AndroidManifest.xml, пользователь видит их при установке приложения.
* **Опасные** — разрешения, требующиеся для получения данных, затрагивающих приватные данные пользователя (как запись, так и чтение). Пример таких данных — контакты, файлы на SD-карте (реальной или виртуальной). Когда приложение запрашивает такое разрешение, пользователю показывается диалог, в котором он может предоставить доступ или отказать в нём.

Все опасные разрешения разделены на соответствующие группы разрешений. Например, приложение запрашивает доступ к изменению контактов, и при этом пользователь уже одобрил чтение контактов — в таком случае, система разрешит доступ к изменению контактов без запроса через диалог.

В случае, если пользователь отказывает в разрешении, приложение может запросить его повторно, и диалог покажется снова. При этом в диалоге будет присутствовать галочка, позволяющая больше не спрашивать пользователя об этом разрешении.

Кроме того, пользователь всегда может подтвердить или же отозвать разрешение в настройках.

Давайте запросим разрешение на получение списка файлов.

В первую очередь, добавим соответствующее разрешение в AndroidManifest.xml.

Разрешения добавляются с использованием тэга uses-permission с атрибутом android:name, в котором указывается название разрешения, которое мы хотим получить. Добавляется этот элемент перед тэгом application.

Нам нужно разрешение android.permission.READ\_EXTERNAL\_STORAGE, поэтому наш манифест будет выглядеть так:

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.test.packages">

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity android:name=".FilePickerActivity" />

</application>

</manifest>

Теперь в FilePickerActivity создайте метод initFileManager():

private void initFileManager() {

}

В нём мы в первую очередь проверим, есть ли у нас разрешение на чтение файлов.

Делается это вызовом метода ContextCompat.checkSelfPermission(), который возвращает PackageManager.PERMISSION\_GRANTED, если пользователь предоставил разрешение, или PackageManager.PERMISSION\_DENIED, если пользователь отказал в нём.

Если разрешения нет, то нужно использовать метод ActivityCompat.requestPermissions(), чтобы запросить разрешение. С помощью этого метода можно запросить несколько разрешений подряд.

Итак, добавьте в FilePickerActivity следующий код:

private void requestPermissions() {

ActivityCompat.requestPermissions(this,

new String[]{Manifest.permission.READ\_EXTERNAL\_STORAGE},

PERMISSION\_REQUEST\_CODE

);

}

В этом методе мы запрашиваем разрешение.

Теперь добавьте такой код в метод initFileManager():

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.READ\_EXTERNAL\_STORAGE) == PackageManager.PERMISSION\_GRANTED) {

// Разрешение предоставлено

fileManager = new FileManager(this);

} else {

requestPermissions();

}

Кроме того, в классе FilePickerActivity нужно создать константу PERMISSION\_REQUEST\_CODE — когда пользователь предоставит разрешение, нам придёт уведомление, и по этому коду мы сможем понять, какое именно разрешение было предоставлено (если мы запрашивали несколько):

private static final int PERMISSION\_REQUEST\_CODE = 1;

А теперь обработаем событие предоставления разрешения. Для этого добавьте следующий метод:

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

}

В этом методе мы должны проверить, предоставил ли пользователь разрешение:

if (requestCode == PERMISSION\_REQUEST\_CODE) {

if (grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

Log.i(TAG, "Permission granted!");

initFileManager();

} else {

Log.i(TAG, "Permission denied");

requestPermissions(); // Запрашиваем ещё раз

}

}

Ну и теперь вызываем initFileManager() в onCreate():

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

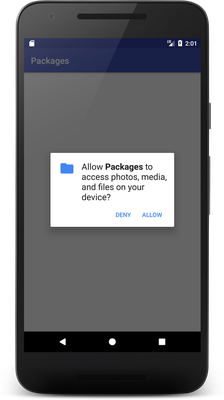
super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_file\_picker);

initFileManager();

}

Теперь запустите приложение и откройте экран выбора файла:

[](https://skillberg.com/media/uploads/2017/11/17/b45d003c819445c9afa6f0306d13dbe2.png)Запрос разрешения

Если вы нажмёте на кнопку **Deny**, диалог покажется ещё раз, и будет показываться до тех пор, пока пользователь не нажмёт **Allow** или не запретит запрашивать разрешение.

На самом деле, бесконечно запрашивать разрешение — это не очень хорошая практика. В случае, если пользователь не подтвердил доступ с первого раза, нужно показать ему информацию о том, зачем нам нужно это разрешение, и что приложение не будет без него корректно работать, после чего предложить нажать на кнопку, чтобы показать диалог ещё раз.

Но для наших нужд в данный момент это не нужно, поэтому оставим пока что такой вариант.

FileManager

Теперь создадим класс, который будет ответственен за получение списка файлов. В нём должна храниться текущая директория, а так же должна быть возможность получения списка файлов в текущей директории с возможностью сортировки.

Создайте новый класс FileManager:

public class FileManager {

}

Объявите в нём поле currentDirectory:

private File currentDirectory;

Так же пригодится поле rootDirectory, в котором мы укажем директорию, выше которой подниматься будет нельзя (это будет та же директория, что и currentDirectory при создании):

private final File rootDirectory;

Теперь создайте конструктор. В конструкторе мы должны проинициализировать currentDirectory.

По умолчанию это будет корень SD-карты. Однако, если SD-карта недоступна, то будем использовать директорию приложения. Для доступа к директории приложения нам понадобится контекст.

Конструктор в итоге будет выглядеть вот так:

public FileManager(Context context) {

File directory;

if (Environment.MEDIA\_MOUNTED.equals(Environment.getExternalStorageState())) {

directory = Environment.getExternalStorageDirectory();

} else {

directory = ContextCompat.getDataDir(context);

}

rootDirectory = directory;

navigateTo(directory);

}

Сперва мы проверяем, доступна ли SD-карта. Если доступна, то получаем её, используя метод Environment.getExternalStorageDirectory(). В ином случае получаем путь к директории данных нашего приложения (в нашем случае это будет /data/data/com.test.packages/).

**Обратите внимание**: несмотря на то, что в большинстве случаев SD-карта доступна по пути /sdcard/, ни в коем случае нельзя использовать эту строковую константу. Вместо этого нужно всегда использоватьEnvironment.getExternalStorageDirectory(), иначе может получиться так, что доступа к карте вы не получите! То же самое справедливо и для директории данных, которую мы используем, если карта недоступна.

В последней строчке мы переходим в нужную нам директорию, используя метод navigateTo(). Этот метод мы ещё не создали — самое время сделать это:

public boolean navigateTo(File directory) {

// Проверим, является ли файл директорией

if (!directory.isDirectory()) {

Log.e(TAG, directory.getAbsolutePath() + " is not a directory!");

return false;

}

// Проверим, не поднялись ли мы выше rootDirectory

if (!directory.equals(rootDirectory) &&

rootDirectory.getAbsolutePath().contains(directory.getAbsolutePath())) {

Log.w(TAG, "Trying to navigate upper than root directory to " + directory.getAbsolutePath());

return false;

}

currentDirectory = directory;

return true;

}

Отлично! Теперь нужно создать метод для навигации в директорию, которая находится выше. Пользователь будет попадать в неё, нажимая кнопку **назад**:

public boolean navigateUp() {

return navigateTo(currentDirectory.getParentFile());

}

Ну и напоследок — метод, с помощью которого мы будем получать список файлов в текущей директории:

public List<File> getFiles() {

List<File> files = new ArrayList<>();

files.addAll(Arrays.asList(currentDirectory.listFiles()));

return files;

}

На данный момент может показаться, что этот метод избыточен, но позже вы поймёте, что мы создали его не просто так :)

Отображаем список файлов

Пришло время отобразить список файлов. Процесс почти ничем не отличается от того, что мы делали ранее с установленными приложениями.

В классе FilePickerActivity создаём поле FileManager:

private FileManager fileManager;

Инициализируем его в методе initFileManager():

private void initFileManager() {

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.READ\_EXTERNAL\_STORAGE) == PackageManager.PERMISSION\_GRANTED) {

// Разрешение предоставлено

fileManager = new FileManager(this);

} else {

requestPermissions();

}

}

Создаём файл лэйаута для элемента ячейки view\_item\_file.xml:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal"

android:paddingBottom="8dp"

android:paddingEnd="8dp"

android:paddingStart="16dp"

android:paddingTop="8dp">

<ImageView

android:id="@+id/icon\_iv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<TextView

android:id="@+id/name\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="8dp" />

</LinearLayout>

На самом деле, тут можно использовать Compound Drawable для оптимизации, но этот способ нам не очень поможет, если мы захотим добавить ещё какую-то информацию о файле, так что пока что оставляем так.

Теперь создаём адаптер:

public class FilesAdapter extends RecyclerView.Adapter<FilesAdapter.ViewHolder> {

private List<File> files = new ArrayList<>();

public void setFiles(List<File> files) {

this.files = files;

}

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

LayoutInflater layoutInflater = LayoutInflater.from(parent.getContext());

View view = layoutInflater.inflate(R.layout.view\_item\_file, parent, false);

return new ViewHolder(view);

}

@Override

public void onBindViewHolder(ViewHolder holder, int position) {

File file = files.get(position);

holder.nameTv.setText(file.getName());

}

@Override

public int getItemCount() {

return files.size();

}

class ViewHolder extends RecyclerView.ViewHolder {

private final TextView nameTv;

public ViewHolder(View itemView) {

super(itemView);

nameTv = itemView.findViewById(R.id.name\_tv);

}

}

}

Создаём инстанс адаптера в методе onCreate() FilePickerActivity, и устанавливаем его в RecyclerView:

private FilesAdapter filesAdapter;

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_file\_picker);

RecyclerView recyclerView = findViewById(R.id.files\_rv);

recyclerView.setLayoutManager(new LinearLayoutManager(this));

filesAdapter = new FilesAdapter();

recyclerView.setAdapter(filesAdapter);

initFileManager();

}

Добавляем метод updateFileList(), который будет обновлять список файлов и передавать его в адаптер:

private void updateFileList() {

List<File> files = fileManager.getFiles();

filesAdapter.setFiles(files);

filesAdapter.notifyDataSetChanged();

}

И вызываем его в методе initFileManager():

private void initFileManager() {

if (ContextCompat.checkSelfPermission(this,

Manifest.permission.READ\_EXTERNAL\_STORAGE) == PackageManager.PERMISSION\_GRANTED) {

// Разрешение предоставлено

fileManager = new FileManager(this);

updateFileList();

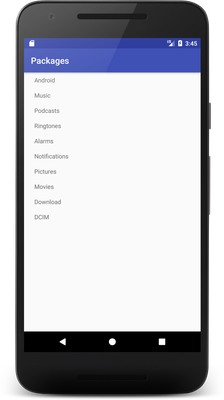
} else {

requestPermissions();

}

}

Запускаем приложение, переходим на экран File Picker'а:

[](https://skillberg.com/media/uploads/2017/11/17/620e2705f7644c519cb5b3c83d4420de.png)Список файлов в RecyclerView

Готово! А в следующем уроке мы научим RecyclerView отличать директории от файлов и начнём обрабатывать нажатия на элементы списка.

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-16)

ДОМАШНЕЕ ЗАДАНИЕ

Добавьте разделители в список.