1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «**Фоновый сервис для ОС Android**»
2. по дисциплине «Введение в мобильные технологии и средства связи»

Вариант 4,4

1. Выполнил
2. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Мясников А.В.

<*подпись*>

1. Санкт-Петербург
2. 2020

**Цели**

Разработать приложение для ОС Google Android, представляющее собой фоновый сервис. Данный сервис должен осуществлять периодический сбор контактов телефонной книги и их выгрузку через облачный сервис Dropbox в соответствии с выданным вариантом.

Выгрузка информации должна осуществляться при наличие любого подключения к сети Интернет и при любом состоянии телефона

**Задачи**

1. Создать в Dropbox в консоли приложений своё приложение.
2. Считывать контакты телефонной книги в файл на SD карту.
3. Сгенерировать токен для доступа к своему Dropbox приложению.
4. Обернуть передачу контактов с SD-карты в Dropbox в сервис

**Ход работы**

Сначала необходимо создать аккаунт на Dropbox и затем создать в консоли приложений своё приложение. Для связи с серверами Dropbox используется API v2:

|  |
| --- |
|  |

Рисунок 1 – Собственное приложение-папка на Dropbox

Далее выставим необходимые разрешения:

|  |
| --- |
|  |

Рисунок 2 – Разрешения

Так как используется интерфейс API v2, то возможно получить доступ не только с помощью двух ключей (OAuth 1), но и второй способ – OAuth2. Второй способ предполагает использование токена, который можно сгенерировать, нажав на кнопку Generate как на рисунке ниже:

|  |
| --- |
|  |

Рисунок 3 – Генерация токена для доступа.

Затем разрешим доступ к приложению, текущий лимит установлен в 500 разработчиков:

|  |
| --- |
|  |

Рисунок 4 – получен доступ к приложению 1/500 пользователей.

Чтобы иметь возможность использовать в своём приложении API Dropbox, необходимо в gradle:build в dependencies выставить следующую настройку:

implementation **'com.dropbox.core:dropbox-core-sdk:3.1.3'**

Для получения доступа был создан класс DropboxClientFactory который инициализирует пользователя сгенерированным токеном, а так же имеет геттер для получения этого пользователя.

Для получения доступа к контактам, интернету и SD-карте были использованы следующие разрешения:

|  |
| --- |
| <**uses-permission android:name="android.permission.INTERNET"** /> <**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** /> <**uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"** /> <**uses-permission android:name="android.permission.READ\_CONTACTS"** /> <**uses-permission android:name="android.permission.WRITE\_CONTACTS"** /> |

Для получения контактов была использована функция getContentResolver. Она возвращает курсор, с помощью которого совершается запрос в базу данных с контактами. Далее идём итератором по колонкам имени и номера. Затем сохраняем их в список, который будет сохранен .txt файлом на SD-карту.

Для того чтобы сделать новый Activity работал как сервис, необходимо выставить в AndroidManifest.xml соответсвующйю настройку:

|  |
| --- |
| <**service android:name=".mine\_service"  android:enabled="true"  android:exported="true"**> </**service**> |

Далее был создан сервис, унаследованный от Service. Сервис запускается и останавливается из MainActivity функциями startService и stopService соответственно.

Для того чтобы сервис периодически опрашивал контакты и отсылал данные в Dropbox, был создан поток класса Runnable, который перестанет это делать, если будет вызван stopService, который в свою очередь вызовет onDestroy, где поток и сервис будет остановлен.

Для отправки текстового файла используется функция:

|  |
| --- |
| mDbxClient.files().uploadBuilder( "/" + remoteFileName)  .withMode(WriteMode.OVERWRITE)  .uploadAndFinish(inputStream); |



Рисунок 5 – Интерфейс запуска и остановки сервиса.

|  |
| --- |
|  |

Рисунок 6 – пришедшие контакты

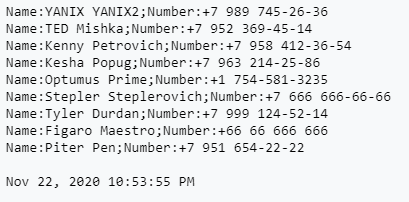


Рисунок 7 – формат записи контактов

**Вывод**

В этой лабораторной работе был создан сервис, который работает в фоновом режиме и отсылает в Dropbox список контактов. Для этого был освоен интерфейс для работы с API Dropbox, создание собственного приложения на AppConsole. А также принцип работы сервисов на Android.

**Листинг**

MainActivity

**package** com.example.imageuploaddropbox;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.core.app.ActivityCompat;  
  
**import** android.Manifest;  
**import** android.content.Intent;  
**import** android.content.pm.PackageManager;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.Toast;  
  
**public class** MainActivity **extends** AppCompatActivity  
{  
  
  
 Button **Start\_service**;  
 Button **Stop\_service**;  
  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 checkPermissionGranted();  
 **Start\_service** = findViewById(R.id.***Start***);  
 **Stop\_service** = findViewById(R.id.***stop***);  
  
 **Start\_service**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 startService(**new** Intent(MainActivity.**this**, mine\_service.**class**));  
 }  
 });  
  
  
 **Stop\_service**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 stopService(**new** Intent(MainActivity.**this**, mine\_service.**class**));  
 Toast.*makeText*(getApplicationContext(), **"SERVICE Successfully STOPPED."**, Toast.***LENGTH\_SHORT***)  
 .show();  
  
 }  
 });  
 }  
  
 **private void** checkPermissionGranted() {  
 **if**(( ActivityCompat.*checkSelfPermission*(getApplicationContext() , Manifest.permission.***READ\_EXTERNAL\_STORAGE*** )  
 != PackageManager.***PERMISSION\_GRANTED*** ) && ( ActivityCompat.*checkSelfPermission*(getApplicationContext() , Manifest.permission.***WRITE\_EXTERNAL\_STORAGE*** )  
 != PackageManager.***PERMISSION\_GRANTED*** )) {  
 Toast.*makeText*(**this**, **"Permission Not Granted"**, Toast.***LENGTH\_SHORT***).show();  
 requestPermission();  
 }  
 **else** {  
 Toast.*makeText*(**this**, **"Permission Granted"**, Toast.***LENGTH\_SHORT***).show();  
  
 }  
 }  
  
 **private void** requestPermission() {  
 ActivityCompat.*requestPermissions*(**this**,  
 **new** String[]{Manifest.permission.***READ\_EXTERNAL\_STORAGE***,  
 Manifest.permission.***WRITE\_EXTERNAL\_STORAGE***} ,123);  
 }  
  
  
  
}

mine\_service

**package** com.example.imageuploaddropbox;  
  
**import** androidx.annotation.Nullable;  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.core.app.ActivityCompat;  
  
**import** android.Manifest;  
**import** android.content.ContentResolver;  
**import** android.content.pm.PackageManager;  
**import** android.database.Cursor;  
**import** android.os.Bundle;  
**import** android.app.Service;  
**import** android.content.Intent;  
**import** android.os.Environment;  
**import** android.os.IBinder;  
**import** android.provider.ContactsContract;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.Toast;  
  
**import** com.dropbox.core.v2.DbxClientV2;  
**import** com.dropbox.core.v2.files.FileMetadata;  
  
**import** java.io.BufferedWriter;  
**import** java.io.File;  
**import** java.io.FileWriter;  
**import** java.io.IOException;  
**import** java.text.DateFormat;  
**import** java.util.Date;  
  
**public class** mine\_service **extends** Service {  
  
  
  
*// class DBThread extends Thread  
// {  
// //Upload c\_up;  
// DBThread()  
// {  
// //this.txt = TEXT1;  
// }  
//  
// @Override  
// public void run()  
// {  
//  
// while (true)  
// {  
//  
// if (Thread.interrupted())  
// {  
// stopSelf();  
//  
// // We've been interrupted: no more crunching.  
// return;  
// }  
//  
// getContactList();  
// uploadFile(uri\_string);  
// try {  
// sleep(5000);  
// //sleep(10000);  
//  
// } catch (InterruptedException e) {  
// // this part is executed when an exception (in this example InterruptedException) occurs  
// }  
// //Toast.makeText(MyService.this, "Thread awakes...",Toast.LENGTH\_SHORT).show();  
// }  
// }  
// }  
//  
// DBThread up=null;* MyRunnable **myRunnable** = **new** MyRunnable();  
  
  
 **public class** MyRunnable **implements** Runnable  
 {  
  
 **private boolean doStop** = **false**;  
  
 **public synchronized void** doStop() {  
 **this**.**doStop** = **true**;  
 }  
  
 **private synchronized boolean** keepRunning() {  
 **return this**.**doStop** == **false**;  
 }  
  
 @Override  
 **public void** run() {  
 **while**(keepRunning())  
 {  
 *// keep doing what this thread should do.* System.***out***.println(**"Running"**);  
 getContactList();  
 uploadFile(**uri\_string**);  
 **try** {  
 Thread.*sleep*(5000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
 }  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 **private** String **ACCESS\_TOKEN** = **"jFeYNiCjICYAAAAAAAAAAY2HSlrD6h1ylieA4A4HjY2TX8A5629z2t-MbQlHMmQh"**;  
 **public static** String *TEXT1* = **""**;  
 **final** String **FILENAME** = **"file"**;  
  
 **final** String **DIR\_SD** = **"MyFiles"**;  
 **final** String **FILENAME\_SD** = **"Contacts.txt"**;  
  
 String **uri\_string**;  
 String **mPath**;  
  
  
 **final** String **LOG\_TAG** = **"myLogs"**;  
  
 @Override  
 **public void** onCreate() {  
 **super**.onCreate();  
 Log.*d*(**LOG\_TAG**, **"onCreate"**);  
 }  
  
 @Override  
 **public int** onStartCommand(Intent intent, **int** flags, **int** startId) {  
 **super**.onStartCommand(intent, flags, startId);  
 Log.*d*(**LOG\_TAG**, **"onStartCommand"**);  
 DropboxClientFactory.*init*(**ACCESS\_TOKEN**);  
 *// MyRunnable myRunnable = new MyRunnable();* Thread thread = **new** Thread(**myRunnable**);  
  
 thread.start();  
 **return** Service.***START\_STICKY***;  
 }  
  
 @Override  
 **public void** onDestroy() {  
 **super**.onDestroy();  
  
 **try** {  
 stopSelf();  
 **myRunnable**.doStop();  
  
 } **catch** (Exception e)  
 {  
 e.printStackTrace();  
 stopSelf();  
 }  
 }  
 @Nullable  
 @Override  
 **public** IBinder onBind(Intent intent) {  
 Log.*d*(**LOG\_TAG**, **"onBind"**);  
 **return null**;  
 }  
  
  
  
  
  
  
  
  
  
 **private void** uploadFile(String uri\_string) {  
 **new** UploadFileTask(**this**, DropboxClientFactory.*getClient*(), **new** UploadFileTask.Callback() {  
 @Override  
 **public void** onUploadComplete(FileMetadata result) {  
 Log.*e*(**"Complete"**, **"onUploadComplete: "** + result.toString() );  
 String message = result.getName() + **" size "** + result.getSize() + **" modified "** +  
 DateFormat.*getDateTimeInstance*().format(result.getClientModified());  
 Toast.*makeText*(getApplicationContext(), message, Toast.***LENGTH\_SHORT***)  
 .show();  
  
  
 Toast.*makeText*(getApplicationContext(), **"Successfully Uploaded."**, Toast.***LENGTH\_SHORT***)  
 .show();  
  
 */\*Intent i = new Intent(getApplicationContext() , GetImageActivity.class);  
 i.putExtra("path\_lower" , result.getPathLower());  
 startActivity(i);\*/* }  
  
 @Override  
 **public void** onError(Exception e) {  
  
 Log.*e*(**"ERROR "**, **"Failed to upload file."**, e);  
  
 }  
 }).execute(uri\_string, **mPath**);  
 }  
  
  
  
  
  
  
  
 **public void** getContactList() {  
 *TEXT1* = **""**;  
 *//requestPermissions(new String[] {Manifest.permission.READ\_CONTACTS}, PERM\_REQUEST);* ContentResolver cr = getContentResolver();  
 Cursor cur = cr.query(ContactsContract.Contacts.***CONTENT\_URI***,  
 **null**, **null**, **null**, **null**);  
  
 **if** ((cur != **null** ? cur.getCount() : 0) > 0) {  
 **while** (cur != **null** && cur.moveToNext()) {  
 String id = cur.getString(  
 cur.getColumnIndex(ContactsContract.Contacts.***\_ID***));  
 String name = cur.getString(cur.getColumnIndex(  
 ContactsContract.Contacts.***DISPLAY\_NAME***));  
  
 **if** (cur.getInt(cur.getColumnIndex(  
 ContactsContract.Contacts.***HAS\_PHONE\_NUMBER***)) > 0) {  
 Cursor pCur = cr.query(  
 ContactsContract.CommonDataKinds.Phone.***CONTENT\_URI***,  
 **null**,  
 ContactsContract.CommonDataKinds.Phone.***CONTACT\_ID*** + **" = ?"**,  
 **new** String[]{id}, **null**);  
 **while** (pCur.moveToNext()) {  
 String phoneNo = pCur.getString(pCur.getColumnIndex(  
 ContactsContract.CommonDataKinds.Phone.***NUMBER***));  
 Log.*i*(**"Srv"**, **"Name: "** + name);  
 *TEXT1* += **"Name:"** + name + **";"**;  
 Log.*i*(**"Srv"**, **"Phone Number: "** + phoneNo);  
 *TEXT1* += **"Number:"**+ phoneNo + **"\n"**;  
 }  
 pCur.close();  
 }  
 }  
 }  
 **if**(cur!=**null**){  
 cur.close();  
 }  
 *//return TEXT;* String currentDateTimeString = DateFormat.*getDateTimeInstance*().format(**new** Date());  
 *TEXT1* += **"\n"** + currentDateTimeString;  
 writeFileSD();  
 }  
  
 **void** writeFileSD() {  
 *// проверяем доступность SD* **if** (!Environment.*getExternalStorageState*().equals(  
 Environment.***MEDIA\_MOUNTED***)) {  
 Log.*d*(**LOG\_TAG**, **"SD-карта не доступна: "** + Environment.*getExternalStorageState*());  
 **return**;  
 }  
 *// получаем путь к SD* File sdPath = Environment.*getExternalStorageDirectory*();  
 *// добавляем свой каталог к пути* sdPath = **new** File(sdPath.getAbsolutePath() + **"/"** + **DIR\_SD**);  
 *// создаем каталог* sdPath.mkdirs();  
 *// формируем объект File, который содержит путь к файлу* File sdFile = **new** File(sdPath, **FILENAME\_SD**);  
 **try** {  
 *// открываем поток для записи* BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(sdFile));  
 *// пишем данные* bw.write(*TEXT1*);  
 *// закрываем поток* bw.close();  
 Log.*d*(**LOG\_TAG**, **"Файл записан на SD: "** + sdFile.getAbsolutePath());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **uri\_string**=sdPath.toString();  
 **mPath**=sdFile.toString();  
 }  
}