МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**ОПТИМИЗАЦИЯ ИСПОЛЬЗОВАНИЯ МОБИЛЬНОГО ТРАФИКА В ОС ANDROID**

Работу выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.С. Глазова   
 (подпись, дата)

Факультет Компьютерных технологий и прикладной математики, курс 3

Специальность/направление 02.03.02. «Фундаментальная математика и информационные технологии»

Научный руководитель, доцент,  
к.т.н, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.А. Приходько  
 (подпись, дата)

Нормоконтролёр, преп,

к.т.н \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е. Полупанова  
 (подпись, дата)

Краснодар 2017

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc483040260)

[1 Постановка задачи 4](#_Toc483040261)

[2 Анализ существующих решений 5](#_Toc483040262)

[2.1 Средства оптимизации, реализованные в программе 9](#_Toc483040263)

[3 Разработка программы 10](#_Toc483040264)

[3.1 Реализация скачивания 12](#_Toc483040265)

[3.2 Реализация кэширования 14](#_Toc483040266)

[4 Тестирование и анализ результатов 16](#_Toc483040267)

[ЗАКЛЮЧЕНИЕ 18](#_Toc483040268)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc483040269)

[ПРИЛОЖЕНИЕ А 20](#_Toc483040270)

[ПРИЛОЖЕНИЕ Б 23](#_Toc483040271)

# ВВЕДЕНИЕ

В наше время люди все чаще используют интернет на мобильных устройствах: для общения, обмена файлами, просмотра видео, прослушивания музыки и т.д. Большинство этих функций требуют больших затрат интернет-трафика, в то время как все мобильные операторы постепенно вводят ограничения на трафик, убирая безлимитные тарифы. Из-за этого остро встает вопрос экономии интернет-трафика. У платформы Android есть собственные способы оптимизации расхода трафика, но зачастую их недостаточно, и разработчикам приложений приходится разрабатывать дополнительные способы решения этой проблемы. Кроме того, некоторые задачи требуют особых алгоритмов решения. Здесь играет важную роль размер файла, его тип и частота использования. Например, есть разница между экономии трафика во время проигрывания небольших музыкальных файлов и больших видеозаписей. Очень важно учитывать все эти нюансы, чтобы получить эффективный и надежный алгоритм.

# 1 Постановка задачи

Задача данной курсовой работы – рассмотреть существующие способы решения задачи экономии мобильного трафика в ОС Android, проанализировать их преимущества и недостатки для некоторых типов данных и, по возможности, найти способы исправления недостатков. Кроме того, реализовать самые эффективные способы оптимизации расхода мобильного интернет-трафика для воспроизведения аудиофайлов в виде программы для ОС Android. Под аудиофайлами подразумевается обычная музыка, которую пользователь будет прослушивать неоднократно.

Для решения задачи экономии мобильного интернет-трафика существует множество алгоритмов, ориентированных на разные типы и размеры данных. Экономия осуществляется путем сокращения объема загружаемых данных, минимизируя количество повторных скачиваний одних и тех же файлов. По статистике, одной из самых затратных по трафику задач является прослушивание музыки, так как большинство пользователей любят прослушивать музыку в свободное время, используя программы для on-line воспроизведения музыки такие как «Apple music» и «Музыка Google». И именно для этой задачи разрабатывается наибольшее количество средств экономии интернет-трафика.

# 2 Анализ существующих решений

Существует множество способов экономии мобильного трафика при прослушивании аудиозаписей, рассмотрим встроенные в ОС Android функции:

* ограничение фонового трафика – многие программы подгружают данные из интернета даже в закрытом состоянии. Зачастую так делают программы для on-line обмена сообщениями и программы, использующие местоположение пользователя (например, погода, карты и т.д.). Обычно загружаются небольшие файлы, но из-за количества таких программ может расходоваться большое количество трафика. Подобная функция помогает пользователю ограничивать расход трафика для программ, которыми он редко пользуется. Но ограничение может вызвать проблемы, связанные с работой ограниченных программ, которые не получили необходимые им данные;
* предупреждение и отключение – в ОС Android также есть функция, позволяющая уведомлять пользователя о расходе трафика и даже отключить мобильный интернет. Выглядит это так: пользователь указывает размер пакета своего мобильного трафика на месяц, после чего система будет оповещать каждый раз, когда количество израсходованного трафика достигнет введенное значение. Кроме того, пользователь может настроить блокировку использования мобильного интернета по достижении введенного значения. Функция очень полезна в ситуациях, когда мобильный оператор автоматически взимает дополнительную плату, если пользователь израсходовал весь доступный трафик раньше времени;
* отключение автоматических обновлений – при желании пользователь может отключить автоматическое обновление программ, что позволит контролировать расход трафика и обновлять программы в удобный для пользователя момент. Это очень полезная функция, но у нее есть один существенный недостаток – вовремя не обновленная программа может работать не корректно, что может вызвать неполадки в работе устройства.

Что касается средств, придуманных разработчиками, то их огромное множество. Рассмотрим самые распространённые из них:

* скачивание на устройство – первое и самое очевидное решение проблемы расхода интернет-трафика, подходящее для всех типов данных. Пользователь загружает нужные файлы на устройство заранее, используя Wi-Fi, что существенно экономит мобильный трафик. Подобная функция имеется у многих программ, например, программа для прослушивания аудиокниг «Патефон» для ОС Android [3]. Но этот способ не всегда удобен. Например, при воспроизведении медиа файлов: пользователю приходится ждать завершения загрузки файла и только потом его воспроизводить. Тем не менее, решение может быть весьма полезным. Некоторые программы используют автоматическую загрузку данных при подключенном Wi-Fi (например, обновление программы), что удобно для пользователя, но порой может вызывать проблемы. Например, если мобильное устройство подключено к Wi-Fi сети с ограниченным трафиком. В таком случае, необходимо спрашивать у пользователя разрешение на загрузку с использованием данной сети. Также к данной функции иногда добавляют предупреждение о размере скачиваемого файла, если пользователь собирается загружать его с помощью мобильного интернета. Это позволяет пользователю контролировать объем загружаемых данных и, если этот объем слишком велик, отложить загрузку до подключения к Wi-Fi сети;
* снижение качества загружаемых данных – функция подходит для медиа файлов и используется в стандартных Google программах (Например, «Музыка Google» для ОС Android [3]). Пользователь может выбрать качество получаемых данных, что напрямую влияет на размер загружаемых файлов: чем хуже качество, тем меньше размер файла. Функция полезна, но страдает качество получаемых данных, из-за чего ее использование имеет значение только в случаях, когда качество файлов не имеет значения;
* сохранение прогресса воспроизведения – одна из самых известных функций, использующаяся в воспроизведении медиа файлов. Отлично подходит для файлов большой длины, но также используется и для небольших файлов. После прекращения воспроизведения записи программа сохраняет место, на котором было прервано воспроизведение и при повторном включении начинает воспроизведение именно с этого места. Это позволяет экономить как время пользователя, так и интернет-трафик, загружая запись с нужного места, а не с самого начала. Подобное решение используется практически во всех программах, связанных с воспроизведением аудиозаписей («Apple Music» для ОС Android [3]) и видеозаписей («YouTube» для ОС Android [3]);
* кэширование файлов – под кэшированием подразумевается сохранение файлов для последующего повторного использования. В случае воспроизведения медиа файлов, кэширование бывает 2 видов: временное сохранение файла в оперативную память (до окончания воспроизведения) и сохранение на постоянной основе в память устройства (до удаления пользователем). Временное сохранение позволяет пользователю перематывать воспроизведение назад и вперед, не загружая каждый раз заново нужный фрагмент файла. По окончании воспроизведения временный файл кэша автоматически удаляется. Кэширование с сохранением на устройстве в основном используется в on-line воспроизведении музыки, в расчете на то, что пользователь будет прослушивать одну и ту же запись несколько раз, что позволяет экономить трафик. Чаще всего такое кэширование происходит таким образом: во время прослушивания аудиозаписи в режиме on-line программа загружает эту запись на устройство для дальнейшего воспроизведения в режиме off-line. При желании пользователь может удалить аудиозапись с устройства. Кэширование аудиозаписей используется во многих популярных on-line плеерах, например, в «ВКонтакте» для ОС Android [3]. Однако, для воспроизведения файлов, не рассчитанных на повторное прослушивание, данный способ не совсем подходит, так как просто будет забивать память устройства ненужными пользователю данными. Данную проблему можно решить таким образом: необходимо сохранять на устройстве лишь часть воспроизводимого файла, начиная воспроизведение после паузы с сохраненного участка, а потом переходя на on-line режим. Куски файла должны перезаписываться на устройство по мере прослушивания пользователем, что сэкономит место на устройстве и мобильный трафик.

2.1 Средства оптимизации, реализованные в программе

В своей программе для on-line воспроизведения аудиозаписей будут реализованы следующие средства экономии мобильного трафика:

* скачивание на устройство – пользователь должен иметь возможность заранее загружать на устройство необходимые ему аудиозаписи. Программа также предусматривает возможность остановки и возобновления загрузки, формирование очереди загрузки и т.д. Кроме того, необходимо присутствие индикатора загрузки и оповещения о конце загрузки;
* кэширование файлов – в программе аудиозаписи будут загружаться и сохраняться на устройство во время воспроизведения, после чего их можно будет прослушать повторно, не используя для этого интернет. При желании пользователь может самостоятельно удалить загруженные файлы из папки загрузки.

Стандартные для ОС Android средства экономии мобильного интернет-трафика пользователь при желании может активировать самостоятельно.

# 3 Разработка программы

Плеер для воспроизведения реализован с помощью класса MediaPlayer [1]. Это стандартный класс Android, имеющий минимальный функционал, но достаточно удобный в реализации. Класс работает на основе состояний, которые управляются с помощью внутреннего конечного автомата – машины состояний. Состояния у класса такие:

* бездействие – объект создан, но воспроизведение пока невозможно;
* инициализация – инициализация источника получения данных;
* подготовка - получение данных для воспроизведения из источника, буферизация и декодирование;
* запущен – начало воспроизведения;
* воспроизведение – процесс воспроизведения;
* приостановлен – воспроизведение поставлено на паузу;
* остановлен – воспроизведение остановлено;
* ошибка – ошибка воспроизведения;
* конец – воспроизведение прекращено.

В результате реализации у плеера имеются кнопки перемотки воспроизведения на 10 секунд и кнопка паузы. Аудиофайлы воспроизводятся и загружаются с сервера «screencast».

На рисунке 1 показан скриншот плеера программы.

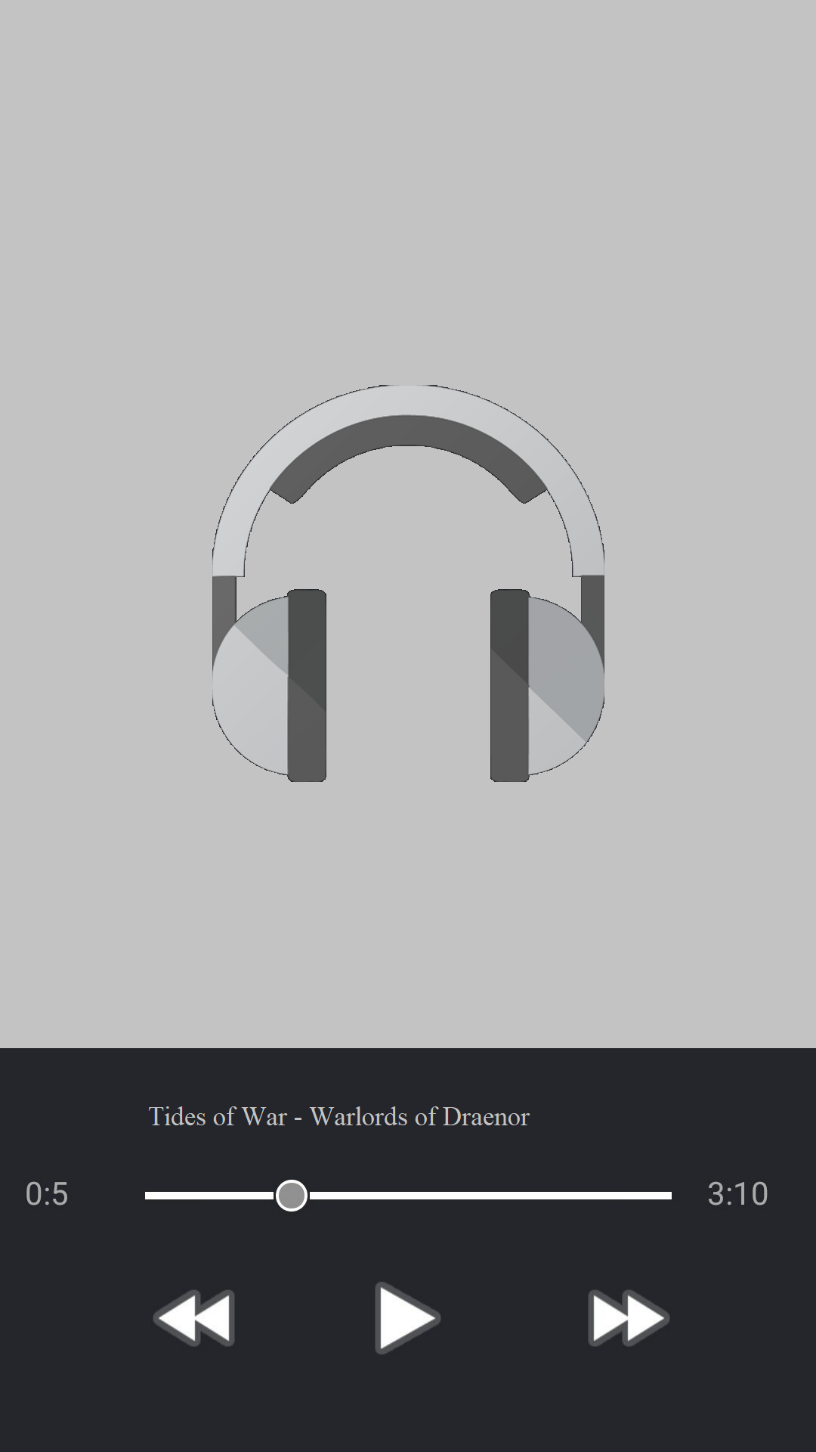


Рисунок 1 – Плеер программы

# 3.1 Реализация скачивания

Для загрузки аудиозаписей использовался класс DownloadManager [1]. Он очень удобен в использовании, так как достаточно указать ссылку на скачивание файла, путь к папке, в которую будет идти загрузка, тип файла и его будущее имя. Кроме того, класс позволяет добавлять оповещения о статусе загрузки.

На рисунке 2 показана схема работы класса DownloadManager.

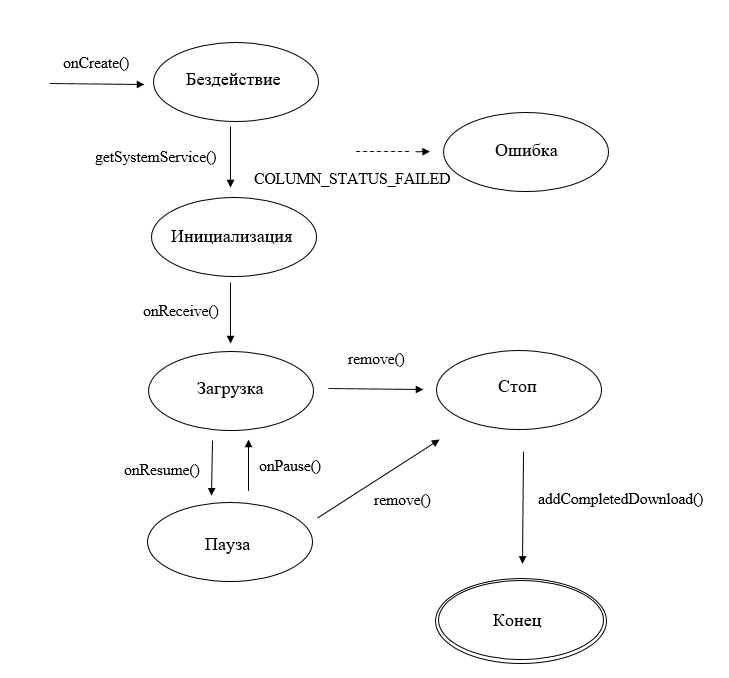


Рисунок 2 – схема работы DownloadManager

После запуска скачивания загрузчик проверяет наличие в очереди других загружающихся файлов и, если такие есть, становится в очередь загрузки и остается в состоянии бездействия до тех пор, пока не придет его очередь. Далее происходит процедура инициализации, где программа устанавливает связь с сервером и получает информацию о будущем местоположении файла на устройстве. Затем начинается загрузка, которая может быть поставлена на паузу или остановлена пользователем. По окончании загрузки процесс скачивания завершается.

Таким образом, программа позволяет скачивать аудиозапись для дальнейшего прослушивания в off-line режиме, что значительно экономит мобильный трафик. Функции паузы и возобновления загрузки служат для обеспечения удобства загрузки.

Код программы с реализацией алгоритма скачивания находится в приложении А.

# 3.2 Реализация кэширования

Кэширование файлов в классе MediaPlayer было реализовано с помощью библиотеки «AndroidVideoCache» [4]. Библиотека была предусмотрена для видео файлов, но мною была переделана для аудио файлов.

На рисунке 3 показана схема воспроизведения с алгоритмом кэширования.

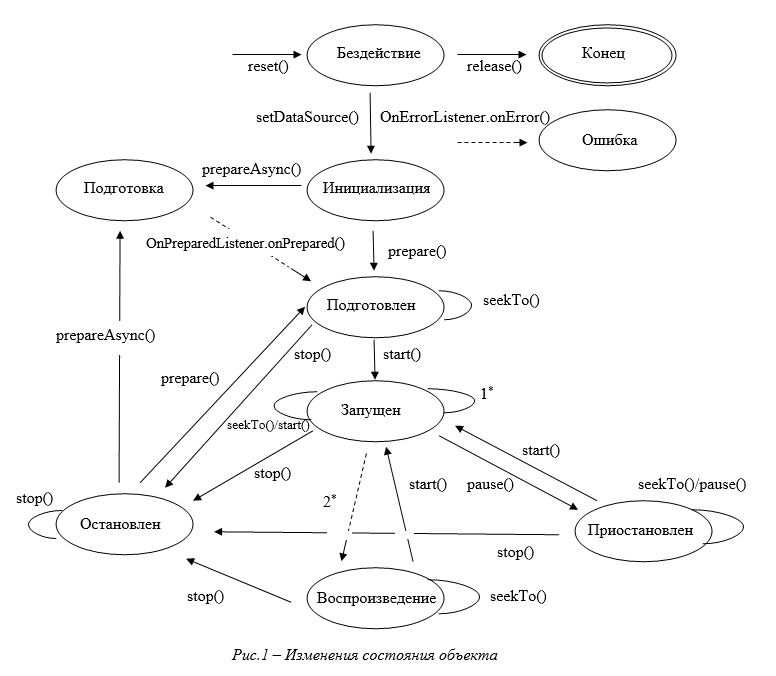


Рисунок 3 – схема воспроизведения с алгоритмом кэширования

1\* - Looping == true && воспроизведение завершается  
2\* - Looping == false && Completion() вызывается для OnCompletionListener.

Из состояния бездействия программа переходит в состояние инициализации, где в первую очередь проверяется наличие кэш-файла текущей аудиозаписи и, если его нет, то устанавливается связь с сервером и памятью устройства для дальнейшего кэширования. Если же такой файл имеется, то воспроизведение идет с устройства без использования интернета, следовательно, соединение с сервером не устанавливается. Далее в состоянии подготовки программа загружает аудиозапись в оперативную память для дальнейшего воспроизведения и сохранения (если идет on-line воспроизведение). Когда часть файла уже загружена, программа переходит в состояние готовности, а из него воспроизведения и начинает проигрывать аудиозапись, продолжая подгружать файл из интернета или внутреннего хранилища устройства. Из состояния воспроизведения по желанию пользователя программа может перейти в состояние паузы, где воспроизведение будет приостановлено, и обратно. При завершении воспроизведения программа переходит в состояние «Стоп», а затем «Конец», освобождая оперативную память и сохраняя файл кэша на устройстве.

Код программы с реализацией алгоритма кэширования находится в приложении Б.

# 4 Тестирование и анализ результатов

Анализ производился для 10 аудиозаписей суммарным объемом равным 112 Мбайт, прослушанных по 5 раз с использованием алгоритма скачивания, кэширования и без какой-либо оптимизации расхода интернет-трафика.

На рисунке 4 показана диаграмма расхода интернет-трафика для каждого варианта воспроизведения. Здесь по горизонтали указываются алгоритмы: без экономии (т.е. без использования средств оптимизации), скачивание на устройство и кэширование с сохранением на устройстве. По вертикали указывается объем загруженных данных. Результаты были получены с помощью стандартного приложения в ОС Android «Использование данных», которое позволяет просмотреть расход мобильного интернет-трафика за определенный период отдельными программами.

Рисунок 4 – результат анализа

Как видно из диаграммы, сохранение файлов, которые не раз будут воспроизводиться пользователем, крайне эффективно, нежели обычное on-line воспроизведение. Это происходит потому, что пользователь загружает файлы лишь 1 раз. Разницы в затратах трафика в алгоритме скачивания и кэширования практически нет, она есть только в удобстве их использования: в случае скачивания пользователю приходится ждать завершения загрузки, чтобы воспроизвести файлы, а в случае кэширования – нет, воспроизведение начинается сразу. Но и кэширование не лишено своих неудобств – оно происходит автоматически и из-за этого пользователь вынужден удалять ненужные ему записи после загрузки.

# ЗАКЛЮЧЕНИЕ

В процессе разработки были проанализированы средства экономии мобильного интернет-трафика и изучены особенности экономии трафика для воспроизведения музыкальных файлов. Также была разработана и протестирована на эффективность программа для on-line воспроизведения аудиофайлов.

В программе реализованы алгоритмы скачивания файлов на устройство и кэширования. У плеера имеются функции паузы и перемотки. Также есть полоса загрузки, позволяющая следить за временем воспроизведения.

Все поставленные задачи выполнены и основные функции программа выполняет. В дальнейшем планируется усовершенствование программы и алгоритмов, реализованных в ней. А также разработка и реализация новых алгоритмов, предназначенных для экономии интернет-трафика.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Developers [Электронный ресурс]. URL: https://developer.android.com/ (дата обращения: 01.12.2016).
2. Давыдов С.В., Ефимов А.А. IntelliJ IDEA. Профессиональное программирование на Java. – СПб.: БХВ-Петербург, 2005. – 800 с.:
3. Google Play [Электронный ресурс]. URL: https://play.google.com/ (дата обращения: 23.03.2017).
4. GitHub [Электронный ресурс]. URL: https:// github.com/ danikula/ AndroidVideoCache (дата обращения: 3.04.17).

# ПРИЛОЖЕНИЕ А

Реализация скачивания на устройство в программе

public class downloadActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) { //инициализация загрузки

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_download);

Boolean down\_check = getIntent().getExtras().getBoolean("Boolean");

if(down\_check) {

downloadManager = (DownloadManager) getSystemService(DOWNLOAD\_SERVICE);

Request request = new Request(Uri.parse(Song\_URL));

request.setTitle(name);

request.setMimeType("application/mp3");

request.setDestinationInExternalPublicDir(Environment.DIRECTORY\_DOWNLOADS, "3.mp3");

request.setNotificationVisibility(DownloadManager.Request.VISIBILITY\_VISIBLE\_NOTIFY\_COMPLETED);

downloadManager.enqueue(request);

}

else {

Intent intent2 = new Intent(this, OfPlayerActivity.class);

intent2.putExtra("Boolean", false);

startActivity(intent2);

}

}

private static final String Song\_URL = url;

DownloadManager downloadManager;

@Override

protected void onResume() { //возобновление загрузки

super.onResume();

registerReceiver(receiver, new IntentFilter(DownloadManager.ACTION\_DOWNLOAD\_COMPLETE));

registerReceiver(receiver, new IntentFilter(DownloadManager.ACTION\_NOTIFICATION\_CLICKED));

@Override

protected void onPause() { //остановка загрузки

super.onPause();

unregisterReceiver(receiver);

}

BroadcastReceiver receiver = new BroadcastReceiver() {

@Override

public void onReceive(Context context, Intent intent) {

String action = intent.getAction();

if (DownloadManager.ACTION\_DOWNLOAD\_COMPLETE.equals(action)){

long downloadId = intent.getLongExtra(DownloadManager.EXTRA\_DOWNLOAD\_ID, 0);

DownloadManager.Query query = new DownloadManager.Query();

query.setFilterById(downloadId);

} else if (DownloadManager.ACTION\_NOTIFICATION\_CLICKED.equals(action)){

long[] ids = intent.getLongArrayExtra(DownloadManager.EXTRA\_NOTIFICATION\_CLICK\_DOWNLOAD\_IDS);

DownloadManager.Query query = new DownloadManager.Query();

query.setFilterById(ids);

}

}

};

# ПРИЛОЖЕНИЕ Б

Реализация кэширования в программе

import android.media.AudioManager;

import android.os.Handler;

import android.os.Message;

import android.support.v4.app.Fragment;

import android.util.Log;

import android.widget.ImageView;

import android.widget.ProgressBar;

import android.media.MediaPlayer;

import android.media.MediaPlayer.OnPreparedListener;

import com.danikula.videocache.CacheListener;

import com.danikula.videocache.HttpProxyCacheServer;

import org.androidannotations.annotations.AfterViews;

import org.androidannotations.annotations.EFragment;

import org.androidannotations.annotations.FragmentArg;

import org.androidannotations.annotations.SeekBarTouchStop;

import org.androidannotations.annotations.ViewById;

import java.io.File;

import java.io.IOException;

@EFragment(R.layout.fragment\_audio)

public class AudioFragment extends Fragment implements CacheListener, OnPreparedListener {

private static final String LOG\_TAG = "AudioFragment";

@FragmentArg String url;

@ViewById ImageView cacheStatusImageView;

MediaPlayer mediaPlayer;

@ViewById ProgressBar progressBar;

Boolean check\_play = false;

private final AudioProgressUpdater updater = new AudioProgressUpdater();

public static Fragment build(String url) {

return AudioFragment\_.builder()

.url(url)

.build();

}

@AfterViews

void afterViewInjected() {

checkCachedState();

startAudio();

}

private void checkCachedState() { //статус кэширования

setCachedState(fullyCached);

if (fullyCached) {

progressBar.setSecondaryProgress(100);

}

}

private void releaseMP() { //инициализация плеера

if (mediaPlayer != null) {

try {

mediaPlayer.release();

mediaPlayer = null;

} catch (Exception e) {

e.printStackTrace();

}

}

}

private void startAudio() { //начало воспроизведения

if (!check\_play) {

releaseMP();

HttpProxyCacheServer proxy = App.getProxy(getActivity());

mediaPlayer = new MediaPlayer();

proxy.registerCacheListener(this, url);

String proxyUrl = proxy.getProxyUrl(url);

Log.d(LOG\_TAG, "Use proxy url " + proxyUrl + " instead of original url " + url);

mediaPlayer.setAudioStreamType(AudioManager.STREAM\_MUSIC);

try {

mediaPlayer.setDataSource(proxyUrl);

mediaPlayer.setOnPreparedListener(this);

mediaPlayer.prepareAsync();

check\_play = true;

} catch (IOException e) {

e.printStackTrace();

}

}

if (mediaPlayer != null) {

if (mediaPlayer.isPlaying()) {

mediaPlayer.pause();

} else {

mediaPlayer.start();

}

}

}

public void onPrepared(MediaPlayer mp) {

mp.start();

}

@Override

public void onResume() { //возобновление воспроизведения

super.onResume();

updater.start();

}

@Override

public void onPause() { //пауза

super.onPause();

updater.stop();

}

@Override

public void onDestroy() { //завершение воспроизведения

super.onDestroy();

mediaPlayer.stop();

App.getProxy(getActivity()).unregisterCacheListener(this);

}

@Override

public void onCacheAvailable(File file, String url, int percentsAvailable) { //проверка на наличие кэш-файла

progressBar.setSecondaryProgress(percentsAvailable);

setCachedState(percentsAvailable == 100);

Log.d(LOG\_TAG, String.format("onCacheAvailable. percents: %d, file: %s, url: %s", percentsAvailable, file, url));

}

private void updateAudioProgress() { //ползунок

int audioProgress = mediaPlayer.getCurrentPosition() \* 100 / mediaPlayer.getDuration();

progressBar.setProgress(audioProgress);

}

@SeekBarTouchStop(R.id.progressBar)

void seekAudio() {

int audioPosition = mediaPlayer.getDuration() \* progressBar.getProgress() / 100;

mediaPlayer.seekTo(audioPosition);

}

private void setCachedState(boolean cached) {

int statusIconId = cached ? R.drawable.ic\_cloud\_done : R.drawable.ic\_cloud\_download;

cacheStatusImageView.setImageResource(statusIconId);

}

private final class AudioProgressUpdater extends Handler {

public void start() {

sendEmptyMessage(0);

}

public void stop() {

removeMessages(0);

}

@Override

public void handleMessage(Message msg) {

updateAudioProgress();

sendEmptyMessageDelayed(0, 500);

}

}

}

private static File getCacheDirectory(Context context, boolean preferExternal) { //путь сохранения кэша

File appCacheDir = null;

String externalStorageState;

try {

externalStorageState = Environment.getExternalStorageState();

} catch (NullPointerException e) {

externalStorageState = "";

}

if (preferExternal && MEDIA\_MOUNTED.equals(externalStorageState)) {

appCacheDir = getExternalCacheDir(context);

}

if (appCacheDir == null) {

appCacheDir = context.getCacheDir();

}

if (appCacheDir == null) {

String cacheDirPath = "/data/data/" + context.getPackageName() + "/cache/";

LOG.warn("Can't define system cache directory! '" + cacheDirPath + "%s' will be used.");

appCacheDir = new File(cacheDirPath);

}

return appCacheDir;

}

private static File getExternalCacheDir(Context context) {

File dataDir = new File(new File(Environment.getExternalStorageDirectory(), "Android"), "data");

File appCacheDir = new File(new File(dataDir, context.getPackageName()), "cache");

if (!appCacheDir.exists()) {

if (!appCacheDir.mkdirs()) {

LOG.warn("Unable to create external cache directory");

return null;

}

}

return appCacheDir;

}

}