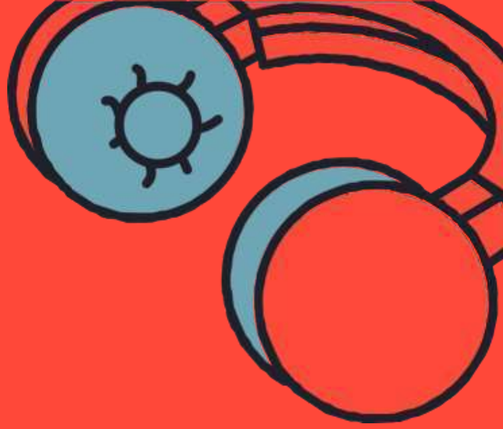
RADIO ROCK

Клиент-серверное ПО



ПРЕКРАСНАЯ МУЗЫКА

ЧТОБЫ РАССЛАБИТЬСЯ И ОТВЛЕЧЬСЯ ОТ ПОВСЕДНЕВНОЙ ЖИЗНИ

НАШЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

ОНЛАЙН ПОТОКОВОЕ ВЕЩАНИЕ С БУФЕРИЗАЦИЕЙ

ЭТО РЕШАЕТ ПРОБЛЕМЫ С ПЛОХИМ КАЧЕСТВОМ ВЕЩАНИЯ

ОПТИМИЗАЦИЯ

ПРОГРАММА НАСТРОЕНА НА ИСПОЛЬЗОВАНИЕ МИНИМАЛЬНОГО КОЛИЧЕСТВА РЕСУРСОВ

ТЕХНИЧЕСКАЯ ЧАСТЬ

1. Асинхронность. В разработке ПО мы заложили асинхронность задач, чтобы мгновенно получить информацию о текущем треке.

2. Модульность и ООП. Мы использовали парадигму ООП для будущих дополнений и новых функций.

3. Ресурсы данных. В программе мы используем разные картинки, иконки и разметки для быстрого и качественного отображения данных.

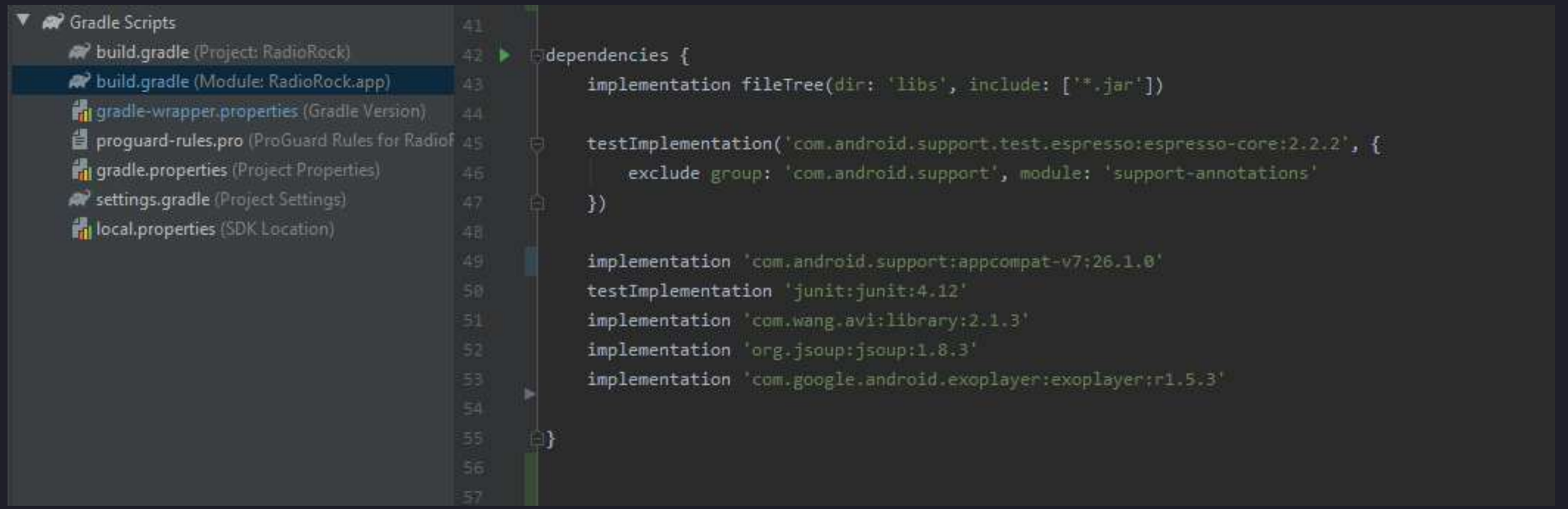
4. Удобная программа для любителей слушать рок-музыку.

Используемые библиотеки

1. Exoplayer
2. AVLoadingIndicatorView
3. CircularSeekBar
4. Jsoup

Библиотеки:

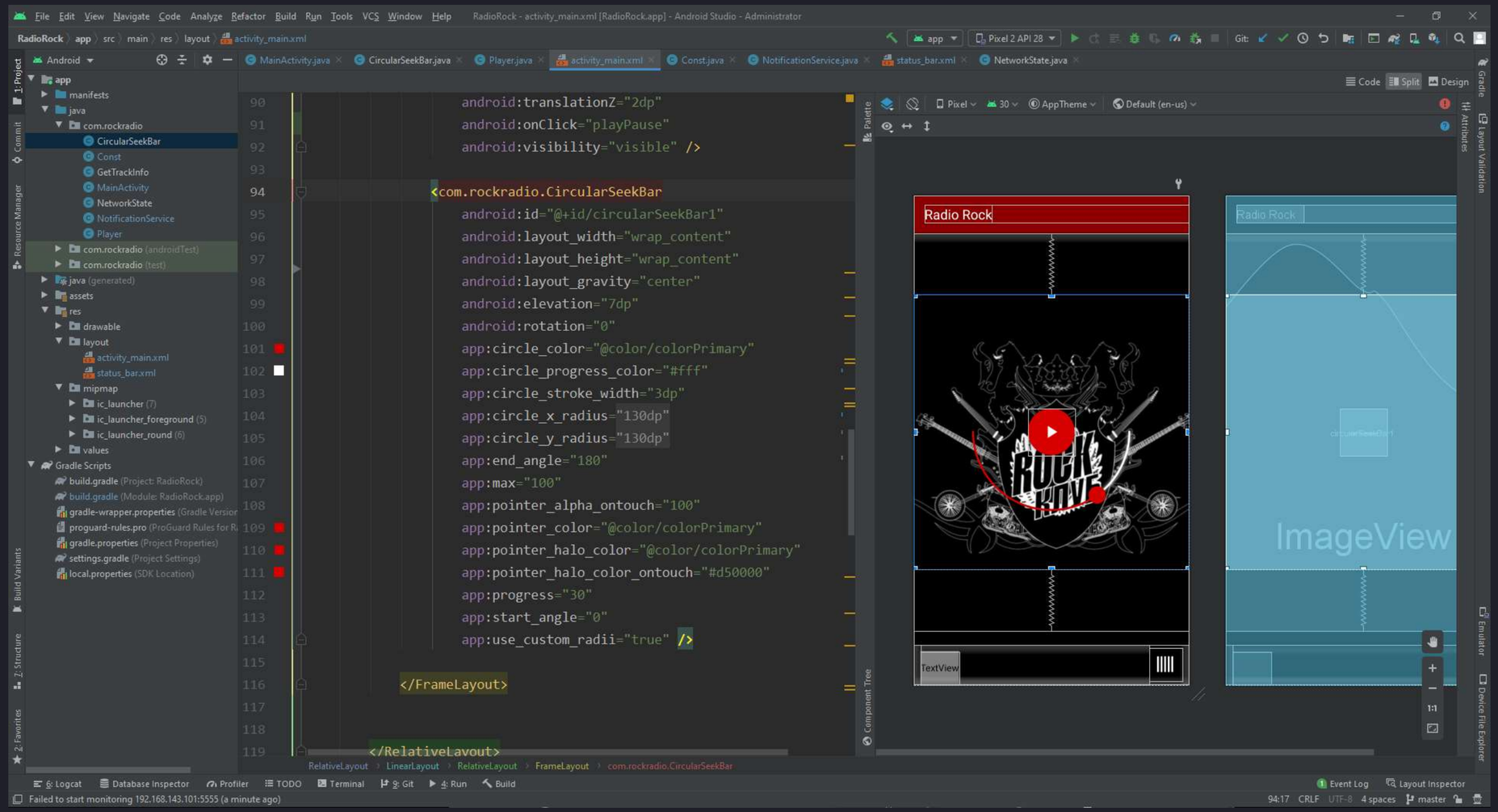
Exoplayer, AVLoadingIndicatorView и Jsoup подключаются в файле сценарии build.gradle (Module: RadioRock.app) как показано в ниже на скриншоте



CircularSeekBar

Используется для кастомного SeekBar и как регулятор громкости.

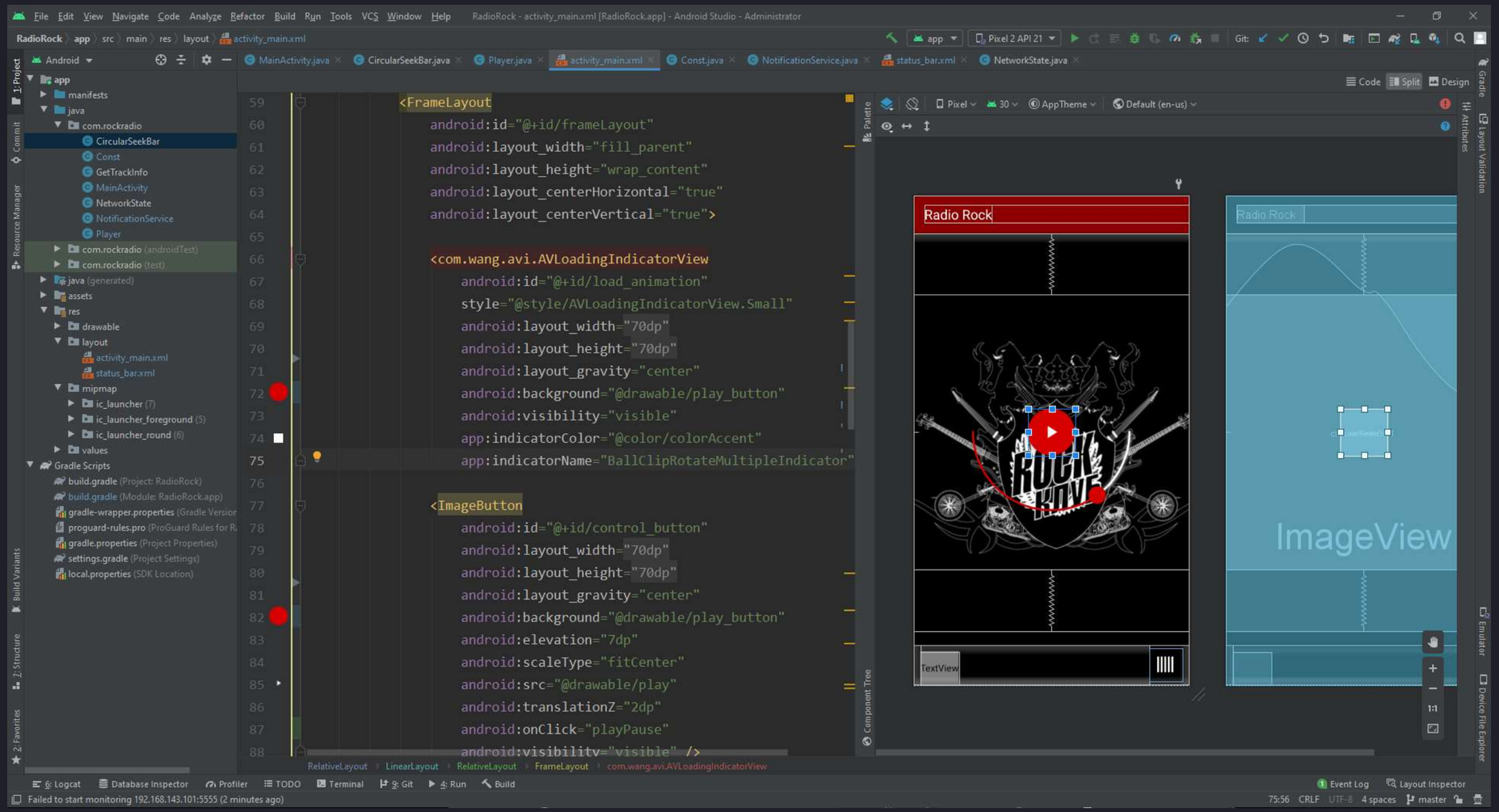
Взята отсюда:
https://github.com/tankery/CircularSeekBar/blob/master/circularSeekBar/src/main/java/me/tankery/lib/circularseekbar/CircularSeekBar.java



AVLoadingIndicatorView

Используется для анимации кнопки PLAY/PAUSE и анимации индикации звучания (см. скриншот).

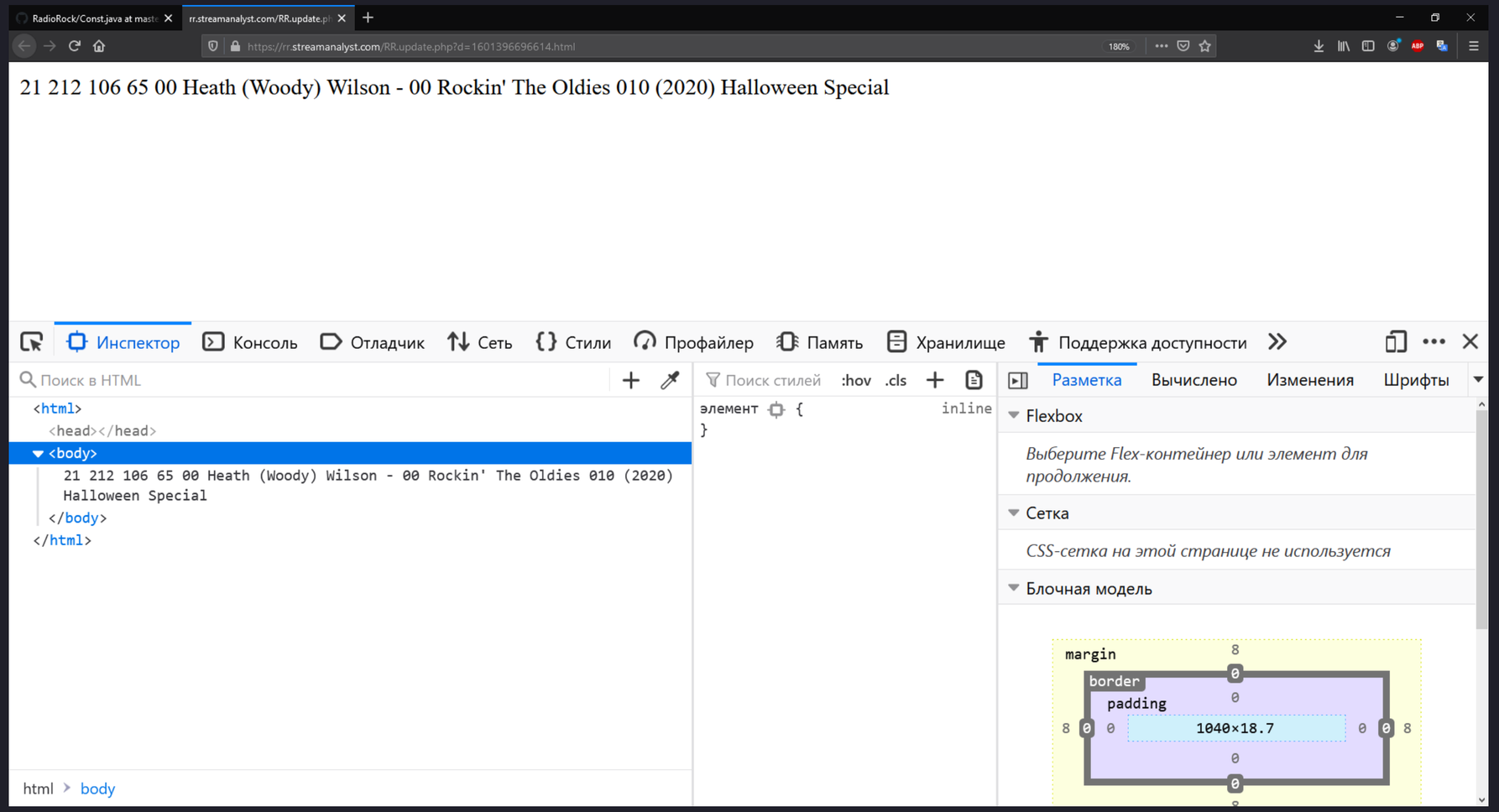
Взята отсюда: https://github.com/81813780/AVLoadingIndicatorView



Jsoup

Используется в классе TrackInfo для парсинга информации о треке в потоковом html файле (см. скриншот)

Взята отсюда: https://jsoup.org/download



```
public static void start(String URL, final Context context)
{
    if(exoPlayer != null)
    {
        exoPlayer.stop();
    }

    // Объявление URI со ссылкой на аудиопоток и извлечение его потокового формата
    Uri URI = Uri.parse(URL);
    FrameworkSampleSource sampleSource = new FrameworkSampleSource(context, URI, null);

    audioRenderer = new MediaCodecAudioTrackRenderer(sampleSource, null, true);

    // инициализация плеера
    exoPlayer = ExoPlayer.Factory.newInstance(1);
    exoPlayer.prepare(audioRenderer);

    // начать проигрывание после окончания буферизации
    exoPlayer.setPlayWhenReady(true);

    // при нажатии на кнопку play
    toast = Toast.makeText(context.getApplicationContext(), "Идет буферизация...",
            Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.BOTTOM, 0, 130);
    toast.show();

    // регистрация состояния плеера
    exoPlayer.addListener(new ExoPlayer.Listener() {
        @Override
        public void onPlayerStateChanged(boolean playWhenReady, int playbackState) {
            if(!isOnline(this))
            {
                toast = Toast.makeText(context.getApplicationContext(), "Нет интернета",
                        Toast.LENGTH_LONG);
                toast.setGravity(Gravity.BOTTOM, 0, 130);
                toast.show();

                Player.stop();
                controlButton.setImageResource(R.drawable.play);
                playingAnimation.setVisibility(View.GONE);

                vibrate(context);
            }
            else
            {
                if(playbackState == STATE_READY)
                {
                    playingAnimation.setVisibility(View.VISIBLE);
                    MainActivity.loadingAnimation.setVisibility(View.GONE);
                    controlButton.setVisibility(View.VISIBLE);
                    controlButton.setImageResource(R.drawable.pause);
                }
            }
        }

        // когда все готово для воспроизведения информируем
        @Override
        public void onPlayWhenReadyCommitted() {
            toast = Toast.makeText(context.getApplicationContext(), "Все готово, Вы можете слушать...",
                    Toast.LENGTH_LONG);
            toast.setGravity(Gravity.BOTTOM, 0, 130);
            toast.show();
        }

        @Override
        public void onPlayerError(ExoPlaybackException error) {
```

Exoplayer

Сам плеер реализован в классе Player (см. в самом проекте). Почему не Media Player? Потому что Media Player на разных устройствах вел себя по разному. Выражалась проблема в зависании, отставании и «прыганьем». Дело было в Media Player или в кривизне дуги земной…

P.S. Что выложить на этом слайде не знаю, ибо выложить целиком код нет смысла, да и посмотреть его можно в самом проекте.

Взята отсюда: https://github.com/google/ExoPlayer

Класс TrackInfo

В классе TrackInfo реализована асинхронность, наследуясь от AsyncTask. Это простой механизм для перемещения трудоёмких операций в фоновый поток. Таким образом, наш UI не будет зависать при запросе, получении и парсинге информации о треке.

```
public class TrackInfo extends AsyncTask <Void, Void, Void>
```

Сам парсинг делаем в методе doInBackground**()**, который выполняется в фоновом потоке.

```
@Override
protected Void doInBackground(Void... params) {
    Document doc = null;
    try {
        doc = Jsoup.connect(Const.TRACK_INFO_URL).get();
    } catch (IOException e) {
        e.printStackTrace();
    }

    firstInfo = doc.select("body").text();

    return null;
}
```

После выполнения фоновой работы вызывается метод onPostExecute(), в который передается результат работы метода doInBackGround()

```
@Override
protected void onPostExecute(Void result) {
    super.onPostExecute(result);
    infoTrack =  firstInfo.replaceAll("[0-9]+", "");
    infoTrack =  infoTrack.replaceAll("[()]+", "");
    MainActivity.setSongData();
}
```

Класс Const

В классе Const определены:

1. Ссылка на сам аудиопоток

```
public static final String RADIO_PATH =
        "http://lin3.ash.fast-serv.com:6026/stream_96";
```

2. Ссылка на страницу с информацией о треке

```
public static final String TRACK_INFO_URL =
        "https://rr.streamanalyst.com/RR.update.php?d=1601396696614.html";
```

3. Время для сна созданного нами потока, который дает «отсрочку» на бездействие

```
public static final int INFO_LOAD_REFRESH_TIME = 300000;
```

4. Время для вибрации

```
public final static int VIBRATE_TIME = 5
```

5. Интерфейс ACTION для намерений уведомлений в классе NotificationService

```
public interface ACTION {
    String MAIN_ACTION = "MAIN_ACTION";
    String PLAY_ACTION = "PLAY_ACTION";
    String STARTFOREGROUND_ACTION = "STARTFOREGROUND_ACTION";
    String STOPFOREGROUND_ACTION = "STOPFOREGROUND_ACTION";
}
```

6. ID уведомления

```
public static int FOREGROUND_SERVICE = 101;
```

Класс NetworkState

В классе используется метод isOnline(), а также перезагруженные методы. Используется для мониторинга сети.

```
public static boolean isOnline()
{
    ConnectivityManager cm =
            (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting())
    {
        return true;
    }
    return false;
}

public static boolean isOnline(Context context)
{
    ConnectivityManager cm =
            (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting())
    {
        return true;
    }
    return false;
}

public static boolean isOnline(ExoPlayer.Listener listener)
{
    ConnectivityManager cm =
            (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting())
    {
        return true;
    }
    return false;
}
```

p.s. понимаю, что должен использовать Broadcast для этой цели, но как-то не получилось, видимо, кривизна дуги земной помешала…
Уверен, что в будущем эта дуга выпрямится и станет лучом в светлое будущее :)

```
private void showNotification(int pos) {
    RemoteViews views = new RemoteViews(getPackageName(), R.layout.status_bar);

    Intent notificationIntent = new Intent(this, MainActivity.class);
    notificationIntent.setAction(Const.ACTION.MAIN_ACTION);
    notificationIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
            | Intent.FLAG_ACTIVITY_CLEAR_TASK);

    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
            notificationIntent, 0);

    Intent playIntent = new Intent(this, NotificationService.class);
    playIntent.setAction(Const.ACTION.PLAY_ACTION);
    PendingIntent pplayIntent = PendingIntent.getService(this, 0,
            playIntent, 0);

    Intent closeIntent = new Intent(this, NotificationService.class);
    closeIntent.setAction(Const.ACTION.STOPFOREGROUND_ACTION);
    PendingIntent pcloseIntent = PendingIntent.getService(this, 0,
            closeIntent, 0);

    views.setOnClickPendingIntent(R.id.status_bar_play, pplayIntent);

    views.setOnClickPendingIntent(R.id.status_bar_collapse, pcloseIntent);


    // состояние после первого нажатия на кнопку play
    if (pos == 0)
    {
        views.setImageViewResource(R.id.status_bar_play, R.drawable.pause_ntf);
    }

    // pos = 1 и pos = 2 состояние после нажатия на кнопку play/pause
    if(pos == 1) {
        views.setImageViewResource(R.id.status_bar_play, R.drawable.pause_ntf);
        if(MainActivity.controlButton != null)
        {
            MainActivity.controlButton.setImageResource(R.drawable.play);
            MainActivity.playingAnimation.setVisibility(View.GONE);
            MainActivity.loadingAnimation.setVisibility(View.VISIBLE);
            MainActivity.controlButton.setVisibility(View.GONE);
            MainActivity.controlIsActivated = true;
        }
    }

    if(pos == 2)
    {
        views.setImageViewResource(R.id.status_bar_play, R.drawable.play_ntf);
        if(MainActivity.controlButton != null)
        {
            MainActivity.controlButton.setImageResource(R.drawable.play);
            MainActivity.playingAnimation.setVisibility(View.GONE);
            MainActivity.loadingAnimation.setVisibility(View.GONE);
            MainActivity.controlButton.setVisibility(View.VISIBLE);
            MainActivity.controlIsActivated = false;
        }
    }

    notification = new Notification.Builder(this).setContentText(infoTrack).build();
    notification.contentView = views;
    notification.flags = Notification.FLAG_ONGOING_EVENT;
    notification.icon = R.drawable.radio;
    notification.contentIntent = pendingIntent;
    startForeground(Const.FOREGROUND_SERVICE, notification);
}
```

Класс NotificationService

В этом классе реализована логика показа уведомлений и робота в фоновом режиме. Работает даже после блокировки телефона. В любом случае на Xiaomi Redmi Note 5 такое прокатывает.

Класс MainActivity

Данный класс является главным и выполняет роль связного для других классов, а также служит для отрисовки UI и инициализации методов и View элементов.

Состоит из методов:

```
// инициализация всех view элементов
@SuppressLint("CutPasteId")
void initialise() {...}

// метод для кастомного шрифта
void setCustomFont() {...}

// метод для прослушивания и изменения громкости от панели поиска до плеера
void startListenVolume() {...}

// метод для установки названия трека и имя исполнителя
public static void setSongData() { infoSong.setText(infoTrack); }

// метод для фонового воспроизведения звука
public void startPlayerService() {...}

// метод для вибрации, если сеть упала/восстановилась
public static void vibrate(Context c) {...}

// Обновление данных о треке
public void startRefreshing()
{...}

@Override
public void onBackPressed() {...}

public void playPause(View view) {...}
```

P.S.

Я прикрепил видео, где демонстрируется компиляция, запуск и эксплуатация на API 21, API 24, API 28, API 30

Поскольку в требованиях сказано, что должна запускаться на ОС Android 5 и выше при ОЗУ от 512 Мб с центральным процессором с одноядерной архитектурой, то для запуска на API 21 будет использована **Genymotion Android Emulator** с настройками