МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра телекоммуникационных систем и вычислительных средств (ТС и ВС)

РЕФЕРАТ
по дисциплине
«Визуальное программирование»

по теме:
СОЗДАНИЕ MP3-ПЛЕЕРА С ПОМОЩЬЮ ПРОГРАММЫ ANDROID STUDIO

Студент:
Группа № ИА-331 *И.А. Иванов*

Предподаватель:
должность, уч. степень, уч. звание *Р.В. Ахпашев*

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках данной работы необходимо создать приложение для воспроизведения аудио формата mp3.

Для этого мы будем использовать компоненты для работы с плеером, представленные в программе **Android Studio**.

1 ТЕОРИЯ

Поподробнее поговорим о том, каким образом будет устроен проект. Приложение предоставляет следующие возможности:

- Воспроизведение аудиофайлов с устройства
- Управление плейлистами
- Отображение метаданных и обложек альбомов
- Система поиска и фильтрации треков

1.1 Переход в PlayerActivity

Переход осуществляется посредством обработки нажатия кнопки из MainActivity

1.2 Работа с музыкой

Для воспроизведения аудио используется класс **MediaPlayer**, который позволяет управлять воспроизведением, перемоткой и другими аспектами аудиофайлов. Для получения списка доступных песен применяется **MediaStore**, который предоставляет доступ к медиатеке устройства.

1.2.1 Кнопки воспроизведения

В приложении есть 3 кнопки для работы с воспроизведением:

- Кнопка **”Play/Pause”** - служит для воспроизведения/паузы трека
- Кнопка **”Next”** - переключает текущий трек на следующий в очереди
- Кнопка **”Previous”** - необходима для воспроизведения предыдущего в очереди трека
- Ползунок - с его помощью можно перематывать трек на необходимую длину

1.2.2 Кнопки для работы с музыкой и плейлистами

- Кнопка **”Выбрать музыку”** - позволяет выбрать трек, содержащийся на телефоне
- Кнопка **”Все песни”** - показывает все треки, из выбранных пользователем
- Кнопка **”Создать плейлист”** - с ее помощью создается плейлист, в котором вводится название и выбираютяя треки, которые в него войдут
- Кнопка **”Мои плейлисты”** - позволяет выбрать плейлист из созданных

1.3 Сохранение данных

Для хранения информации о созданных плейлистах используется механизм сериализации данных в JSON-формат с помощью библиотеки Gson и сохранение их в **SharedPreferences** — легкий механизм хранения настроек и небольших данных.

1.4 Создание плейлиста

При нажатии на кнопку ”создать плейлист”пользователь выбирает, как будет называться плейлист и какие треки будут в него входить

Также пользователь имеет возможность выбирать, какой именно плейлист он хочет открыть.

2 ПРАКТИКА

Далле будет представлена программная реализации описанного ранее

2.1 Основной класс: Player

Это AppCompatActivity, главный экран приложения. Он содержит всю логику работы плеера.

2.1.1 Основные переменные и компоненты UI

MediaPlayer — для воспроизведения аудио:

SeekBar — ползунок для отображения прогресса текущего трека и его управления:

Кнопки: Play/Pause (playButton), Next (nextButton), Previous (prevButton), а также кнопки для выбора музыки, просмотра всех песен, создания плейлистов и просмотра существующих.

TextView: trackName — название текущего трека.

ImageView: albumArt — изображение обложки альбома.

SharedPreferences — для сохранения данных о плейлистах.

Программная реализация:

```
private lateinit var mediaPlayer: MediaPlayer
    private lateinit var seekBar: SeekBar
    private lateinit var playButton: Button
    private lateinit var nextButton: Button
    private lateinit var prevButton: Button
    private lateinit var trackName: TextView
    private lateinit var selectMusicButton: Button
    private lateinit var showSongsButton: Button
    private lateinit var createPlaylistButton: Button
    private lateinit var showPlaylistsButton: Button
    private lateinit var albumArt: ImageView

    private val handler = Handler(Looper.getMainLooper())
```

```
    private var isPlaying = false
    private var currentTrackIndex = 0
    private var tracks = mutableListOf<Track>()
    private var playlists = mutableListOf<Playlist>()
    private lateinit var sharedPrefs: SharedPreferences
```

2.1.2 Данные

Track — класс данных для хранения информации о песне: URI файла, название, ID альбома и длительность.

Playlist — класс данных для хранения названия плейлиста и списка треков.

Программная реализация:

```
data class Track(
        val uri: Uri,
        val name: String,
        val albumId: Long,
        val duration: Long
    )

    data class Playlist(
        val name: String,
        val tracks: MutableList<Track> = mutableListOf()
    )
```

2.2 Инициализация (onCreate)

Устанавливается layout.

Инициализируются UI-компоненты (initViews()).

Настраивается MediaPlayer.

Настраиваются обработчики кнопок (setupButtons()).

Загружаются сохранённые плейлисты из SharedPreferences.

Проверяется разрешение на чтение внешнего хранилища (checkPermissions()).

Программная реализация:

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_player2)

        initViews()
        setupMediaPlayer()
        setupButtons()
        loadPlaylists()
        checkPermissions()
    }

    private fun initViews() {
        seekBar = findViewById(R.id.seekBar)
        playButton = findViewById(R.id.playButton)
        nextButton = findViewById(R.id.nextButton)
        prevButton = findViewById(R.id.prevButton)
        trackName = findViewById(R.id.trackName)
        selectMusicButton = findViewById(R.id.selectMusicButton)
        showSongsButton = findViewById(R.id.showSongsButton)
        createPlaylistButton = findViewById(R.id.createPlaylistButton
        showPlaylistsButton = findViewById(R.id.showPlaylistsButton)
        albumArt = findViewById(R.id.albumArt)
        sharedPrefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE)
        albumArt.setImageResource(R.drawable.ic_music_note)
    }

    private fun setupMediaPlayer() {
        mediaPlayer = MediaPlayer().apply {
            setOnCompletionListener { nextTrack() }
            setOnPreparedListener {
                start()
                this@Player.isPlaying = true
                playButton.text = "Pause"
                startSeekbarUpdate()
            }
```

```
            setOnErrorListener { _, what, extra ->
                Log.e(TAG, "Error what=$what extra=$extra")
                showToast("             ")
                false
            }
        }
    }

    private fun setupButtons() {
        selectMusicButton.setOnClickListener { checkPermissionAndBrow
        showSongsButton.setOnClickListener { showAllSongsDialog() }
        createPlaylistButton.setOnClickListener { showCreatePlaylistD
        showPlaylistsButton.setOnClickListener { showPlaylistsDialog(

        playButton.setOnClickListener {
            if (tracks.isEmpty()) {
                showToast("              ")
                return@setOnClickListener
            }
            if (isPlaying) pauseMusic() else playMusic()
        }

        nextButton.setOnClickListener {
            if (tracks.isEmpty()) {
                showToast("              ")
                return@setOnClickListener
            }
            nextTrack()
        }

        prevButton.setOnClickListener {
            if (tracks.isEmpty()) {
                showToast("              ")
                return@setOnClickListener
            }
```

```
            previousTrack()
        }

        seekBar.setOnSeekBarChangeListener(object : SeekBar.OnSeekBar
            override fun onProgressChanged(seekBar: SeekBar?, progres
                if (fromUser && mediaPlayer.isPlaying) {
                    mediaPlayer.seekTo(progress)
                }
            }
            override fun onStartTrackingTouch(seekBar: SeekBar?) {}
            override fun onStopTrackingTouch(seekBar: SeekBar?) {}
        })
    }

    private fun checkPermissions() {
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
                REQUEST_PERMISSION
            )
        }
    }
```

2.3 Работа с разрешениями

Если разрешение не предоставлено, запрашивается у пользователя. После получения разрешения вызывается loadTracks(), которая загружает все песни из внешнего хранилища.

Программная реализация:

```
override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, gr
        when (requestCode) {
            REQUEST_PERMISSION -> {
                if (grantResults.isNotEmpty() && grantResults[0] == P
                    loadTracks()
                } else {
                    showToast("                          ")
                }
            }
        }
    }

    private fun loadTracks() {
        tracks.addAll(getAllSongsFromStorage())
        if (tracks.isNotEmpty()) {
            playTrack(0)
        }
    }
```

2.4 Получение песен из хранилища

Метод getAllSongsFromStorage.Он собирает список объектов Track.

Также есть метод addTrackFromUri(), который добавляет трек по URI, полученному при выборе файла через диалог.

Программная реализация:

```
private fun getAllSongsFromStorage(): List<Track> {
        val songs = mutableListOf<Track>()
        if (ContextCompat.checkSelfPermission(
                this,
```

```
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            return songs
        }

        val projection = arrayOf(
            MediaStore.Audio.Media._ID,
            MediaStore.Audio.Media.TITLE,
            MediaStore.Audio.Media.ALBUM_ID,
            MediaStore.Audio.Media.DURATION
        )

        contentResolver.query(
            MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
            projection,
            "${MediaStore.Audio.Media.IS_MUSIC} != 0",
            null,
            "${MediaStore.Audio.Media.TITLE} ASC"
        )?.use { cursor ->
            while (cursor.moveToNext()) {
                val id = cursor.getLong(cursor.getColumnIndexOrThrow(
                val name = cursor.getString(cursor.getColumnIndexOrTh
                val albumId = cursor.getLong(cursor.getColumnIndexOrT
                val duration = cursor.getLong(cursor.getColumnIndexOr
                val uri = ContentUris.withAppendedId(MediaStore.Audio
                songs.add(Track(uri, name, albumId, duration))
            }
        }
        return songs
    }

private fun addTrackFromUri(uri: Uri) {
        try {
            val projection = arrayOf(
```

```
                MediaStore.Audio.Media.TITLE,
                MediaStore.Audio.Media.ALBUM_ID,
                MediaStore.Audio.Media.DURATION
            )

            contentResolver.query(uri, projection, null, null, null)?
                if (cursor.moveToFirst()) {
                    val name = cursor.getString(cursor.getColumnIndex
                    val albumId = cursor.getLong(cursor.getColumnInde
                    val duration = cursor.getLong(cursor.getColumnInd
                    tracks.add(Track(uri, name, albumId, duration))
                }
            }
        } catch (e: Exception) {
            Log.e(TAG, "Error adding track from URI", e)
        }
    }
```

2.5 Воспроизведение музыки

Основной метод — playTrack(index). Он:
Сбрасывает текущий MediaPlayer.
Устанавливает источник данных (URI выбранного трека).
Готовит плеер асинхронно (prepareAsync()).
Обновляет название трека и изображение обложки.
После подготовки запускает воспроизведение (start()).
Обложка альбома загружается через ContentUri с ID альбома.

Программная реализация:

```
private fun playTrack(index: Int) {
        if (index < 0 || index >= tracks.size) return

        try {
            mediaPlayer.reset()
            mediaPlayer.setDataSource(applicationContext, tracks[inde
```

```
            mediaPlayer.prepareAsync()
            currentTrackIndex = index
            trackName.text = tracks[index].name
            loadAlbumArt(tracks[index].albumId)
        } catch (e: Exception) {
            Log.e(TAG, "Play track error", e)
            showToast("    : ${e.message}")
        }
    }
```

2.6 РУправление воспроизведением

Кнопки:

Play/Pause: запускает или ставит на паузу текущий трек.

Next/Previous: переключают на следующий или предыдущий трек в списке.

Автоматически при завершении трека вызывается nextTrack(), чтобы перейти к следующему.

Программная реализация:

```
private fun nextTrack() {
        if (tracks.isEmpty()) return
        currentTrackIndex = if (currentTrackIndex < tracks.size - 1)
        playTrack(currentTrackIndex)
    }
```

2.7 Обновление прогресс-бара

Метод startSeekbarUpdate() запускает цикл обновления прогресса каждые 1 секунду через Handler, синхронизируя SeekBar с текущим положением воспроизведения.

Пользователь может перемещать ползунок для перемотки трека.

Программная реализация:

```
 private fun startSeekbarUpdate() {
        handler.post(object : Runnable {
```

```
            override fun run() {
                if (mediaPlayer.isPlaying) {
                    seekBar.progress = mediaPlayer.currentPosition
                    seekBar.max = mediaPlayer.duration
                    handler.postDelayed(this, 1000)
                }
            }
        })
    }
```

2.8 Работа с диалогами

2.8.1 Все песни (showAllSongsDialog())

Показывает список всех песен в виде RecyclerView с адаптером SongAdapter. При выборе песни она добавляется в текущий список воспроизведения и запускается.

Программная реализация:

```
private fun showAllSongsDialog() {
        val dialogView = LayoutInflater.from(this).inflate(R.layout.d
        val recyclerView = dialogView.findViewById<RecyclerView>(R.id
        val adapter = SongAdapter(getAllSongsFromStorage()) { track -
            tracks.clear()
            tracks.add(track)
            currentTrackIndex = 0
            playTrack(currentTrackIndex)
        }

        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = adapter

        AlertDialog.Builder(this)
            .setTitle("     ")
            .setView(dialogView)
```

```
            .setPositiveButton("    ", null)
            .show()
    }
```

2.8.2 Создание плейлиста (showCreatePlaylistDialog())

Позволяет выбрать несколько песен (режим мультивыбора) и задать название плейлиста. После подтверждения создаётся объект Playlist, он сохраняется в списке и сериализуется в SharedPreferences.

Программная реализация:

```
private fun showCreatePlaylistDialog() {
        val dialogView = LayoutInflater.from(this).inflate(R.layout.d
        val playlistNameEditText = dialogView.findViewById<EditText>(
        val songsRecyclerView = dialogView.findViewById<RecyclerView>

        val allSongs = getAllSongsFromStorage()
        val adapter = SongAdapter(allSongs) { }
        adapter.setMultiSelectMode(true)

        songsRecyclerView.layoutManager = LinearLayoutManager(this)
        songsRecyclerView.adapter = adapter

        AlertDialog.Builder(this)
            .setTitle("         ")
            .setView(dialogView)
            .setPositiveButton("    ") { _, _ ->
                val playlistName = playlistNameEditText.text.toString
                if (playlistName.isNotEmpty() && adapter.getSelectedS
                    val newPlaylist = Playlist(playlistName)
                    newPlaylist.tracks.addAll(adapter.getSelectedSong
                    playlists.add(newPlaylist)
                    savePlaylists()
                    showToast("         ")
                } else {
```

```
                showToast("                    ")
            }
        }
        .setNegativeButton("   ", null)
        .show()
}
```

2.8.3 Просмотр плейлистов (showPlaylistsDialog())

Показывает список созданных плейлистов. При клике загружается их содержимое в текущий список воспроизведения и запускается первая песня.

Также реализовано удаление плейлиста по долгому нажатию с подтверждением.

Программная реализация:

```
private fun showPlaylistsDialog() {
        if (playlists.isEmpty()) {
            showToast("               ")
            return
        }

        val dialogView = LayoutInflater.from(this).inflate(R.layout.d
        val recyclerView = dialogView.findViewById<RecyclerView>(R.id

        val adapter = object : RecyclerView.Adapter<PlaylistViewHolde
            override fun onCreateViewHolder(parent: ViewGroup, viewTy
                val view = LayoutInflater.from(parent.context)
                    .inflate(R.layout.item_playlist, parent, false)
                return PlaylistViewHolder(view, this)
            }

            override fun onBindViewHolder(holder: PlaylistViewHolder,
                val playlist = playlists[position]
                holder.bind(playlist) {
                    tracks.clear()
```

```
                    tracks.addAll(playlist.tracks)
                    currentTrackIndex = 0
                    playTrack(currentTrackIndex)
                }
            }

            override fun getItemCount(): Int = playlists.size
        }

        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = adapter

        AlertDialog.Builder(this)
            .setTitle("       ")
            .setView(dialogView)
            .setPositiveButton("    ", null)
            .show()
    }
```

2.9 Сохранение и загрузка плейлистов

Используется Gson для сериализации/десериализации списка плейлистов в JSON строку, которая хранится в SharedPreferences под ключом ”playlists”.

2.10 Дополнительные компоненты

2.10.1 Адаптеры RecyclerView

SongAdapter — отображает список песен с возможностью мультивыбора.

PlaylistViewHolder внутри метода отображения плейлистов — отображает название, количество треков и изображение обложки; поддерживает удаление по долгому нажатию.

Программная реализация:

```
inner class PlaylistViewHolder(
        itemView: View,
        private val adapter: RecyclerView.Adapter<*>
    ) : RecyclerView.ViewHolder(itemView) {
        private val playlistName: TextView = itemView.findViewById(R.
        private val trackCount: TextView = itemView.findViewById(R.id
        private val playlistArt: ImageView = itemView.findViewById(R.

        fun bind(playlist: Playlist, onClick: () -> Unit) {
            playlistName.text = playlist.name
            trackCount.text = "${playlist.tracks.size}    "

            if (playlist.tracks.isNotEmpty()) {
                val albumArtUri = ContentUris.withAppendedId(
                    Uri.parse("content://media/external/audio/albumar
                    playlist.tracks[0].albumId
                )
                playlistArt.setImageURI(albumArtUri)
                if (playlistArt.drawable == null) {
                    playlistArt.setImageResource(R.drawable.ic_music_
                }
            } else {
                playlistArt.setImageResource(R.drawable.ic_music_note
            }

            itemView.setOnClickListener { onClick() }

            itemView.setOnLongClickListener {
                AlertDialog.Builder(itemView.context)
                    .setTitle("         ?")
                    .setMessage("      ,             '${playlist.name
                    .setPositiveButton("    ") { _, _ ->
                        playlists.removeAt(adapterPosition)
                        savePlaylists()
```

```
                        adapter.notifyItemRemoved(adapterPosition)
                        showToast("         ")
                    }
                    .setNegativeButton("   ", null)
                    .show()
                true
            }
        }
    }

class SongAdapter(
    private val songs: List<Player.Track>,
    private val onItemClick: (Player.Track) -> Unit
) : RecyclerView.Adapter<SongAdapter.SongViewHolder>() {

    private val selectedSongs = mutableSetOf<Player.Track>()
    private var isMultiSelectMode = false

    fun setMultiSelectMode(enabled: Boolean) {
        isMultiSelectMode = enabled
        notifyDataSetChanged()
    }

    fun getSelectedSongs(): List<Player.Track> {
        return selectedSongs.toList()
    }

    inner class SongViewHolder(itemView: View) : RecyclerView.ViewHol
        private val songName: TextView = itemView.findViewById(R.id.s
        private val songDuration: TextView = itemView.findViewById(R.
        private val albumArt: ImageView = itemView.findViewById(R.id.

        fun bind(track: Player.Track) {
            songName.text = track.name
            songDuration.text = formatDuration(track.duration)
```

```
            val albumArtUri = ContentUris.withAppendedId(
                Uri.parse("content://media/external/audio/albumart"),
                track.albumId
            )
            albumArt.setImageURI(albumArtUri)
            if (albumArt.drawable == null) {
                albumArt.setImageResource(R.drawable.ic_music_note)
            }

            itemView.isSelected = selectedSongs.contains(track)
            itemView.setOnClickListener {
                if (isMultiSelectMode) {
                    toggleSelection(track)
                } else {
                    onItemClick(track)
                }
            }
        }

        private fun toggleSelection(track: Player.Track) {
            if (selectedSongs.contains(track)) {
                selectedSongs.remove(track)
            } else {
                selectedSongs.add(track)
            }
            notifyItemChanged(adapterPosition)
        }

        private fun formatDuration(duration: Long): String {
            val seconds = (duration / 1000) % 60
            val minutes = (duration / (1000 * 60)) % 60
            return String.format("%02d:%02d", minutes, seconds)
        }
    }
```

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int)
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_song, parent, false)
        return SongViewHolder(view)
    }

    override fun onBindViewHolder(holder: SongViewHolder, position: I
        holder.bind(songs[position])
    }

    override fun getItemCount(): Int = songs.size
}
```

3 ЗАКЛЮЧЕНИЕ

В ходе выполненной работы была создана полноценная мобильная музыкальная приложение для платформы Android на языке Kotlin. Реализован функционал, включающий загрузку и отображение музыкальных файлов из внешнего хранилища, управление воспроизведением (воспроизведение, пауза, переключение треков), отображение информации о текущем треке и его обложки.

Дополнительно реализованы возможности создания пользовательских плейлистов, их сохранения и последующего просмотра и воспроизведения. В приложении использованы современные компоненты Android, такие как RecyclerView для отображения списков, а также реализована работа с разрешениями и сохранением данных через SharedPreferences с использованием Gson.

Работа продемонстрировала навыки интеграции мультимедийных API, работы с пользовательским интерфейсом и хранения данных, что является важным этапом в разработке мобильных приложений. Итоговая реализация обеспечивает удобство использования и расширяемость, что создает хорошую базу для дальнейшего развития проекта или внедрения дополнительных функций.

4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Документация Kotlin: https://kotlinlang.org/docs/home.html
Документация Android Jetpack: https://developer.android.com/jetpack