PODCAST PLUS: A REDUX-INSPIRED PODCAST APP WITH DYNAMIC THEMES FOR ANDROID

ABSTRACT:

A project that demonstrates the use of Android Jetpack Compose to build a UI for a podcast player app. The app allows users to choose , play and pause podcasts.

INTRODUCTION

I have built a basic podcast app with an interactive UI, as well as some of its functionalities :

- Like System
- Multi-word search system.

The app currently has 5 screens :

- Home: With a hard-coded list of podcasts and an integrated search field. The screen adapts automatically depending on whether the TextField is empty or not.
- Favorite: Displays liked podcasts and a button for each podcast to unlike it.
- Podcast: Displays information and the episodes list of a particular podcast.
- Episode: Displays information about the episode, a dynamic slider that changes the duration synchronously, and other non-functional UI Buttons (Inspired from an existing UI).
- About: Displays basic information about the app.

Free React resources are very difficult to find when searching for templates and themes on the Internet.

Even if you don’t care about the quality, they seem pretty undiscoverable, so moved by curiosity, I spent hours digging around on Google and Github, and the result is this nice collection of 35+ free React templates and themes. I wouldn't have bet on it, but they are also high-quality resources.

So, in this list will you find a vast variety of templates and themes to build pretty much anything you can imagine. For example:

- Admin dashboards
- Websites
- Landing pages
- Online portfolios
- Blogs
- Design systems

I’ve not distributed the resources into sections (e.g. website templates) because not everything in this list falls under a specific category (e.g. component libraries, UI Kits, etc..), so I would suggest you browse the entire article and bookmark your favorite ones.
A last point: Even although we can't consider component libraries and UI Kits as templates and themes, they are amazing starting points to kickstart new projects, so I thought it was worth mentioning a few of them in this collection.
Open is a free React template created for developers who want to create a quick and professional landing page for their open source projects, online services, digital products, and more. With an aim to capture leads and email subscribers, Open offers a versatile library of sleek, minimalistic, and reusable components and elements.

Features:

- Designed for open source products and online services
- Dark and minimalistic design
- Fully downloadable via Github

Atomize

Atomize is a React UI framework designed to helps developers cooperate with designers and build consistent and harmonious user interfaces without efforts. Thanks to a perfect combination of resources such as style guides and flexible grids, Atomize is suitable to create any kind of responsive websites.

Features:

- Designed for open source products and online services
- Dark and minimalistic design
- Fully downloadable via Github

Treact

Treact is a gallery of free and modern React templates and UI components developed using TailwindCSS as the front-end framework. This archive of beautiful resources provides 7 pre-built main pages, 8 secondary pages, and 52 pre-designed elements and sections. Every piece of contente is fully customisable and scalable for desktop, tablet, and mobile.

Features:

- Rich gallery of templates and blocks
- Consistent imagery and illustrations
- Modern and versatile look and feel

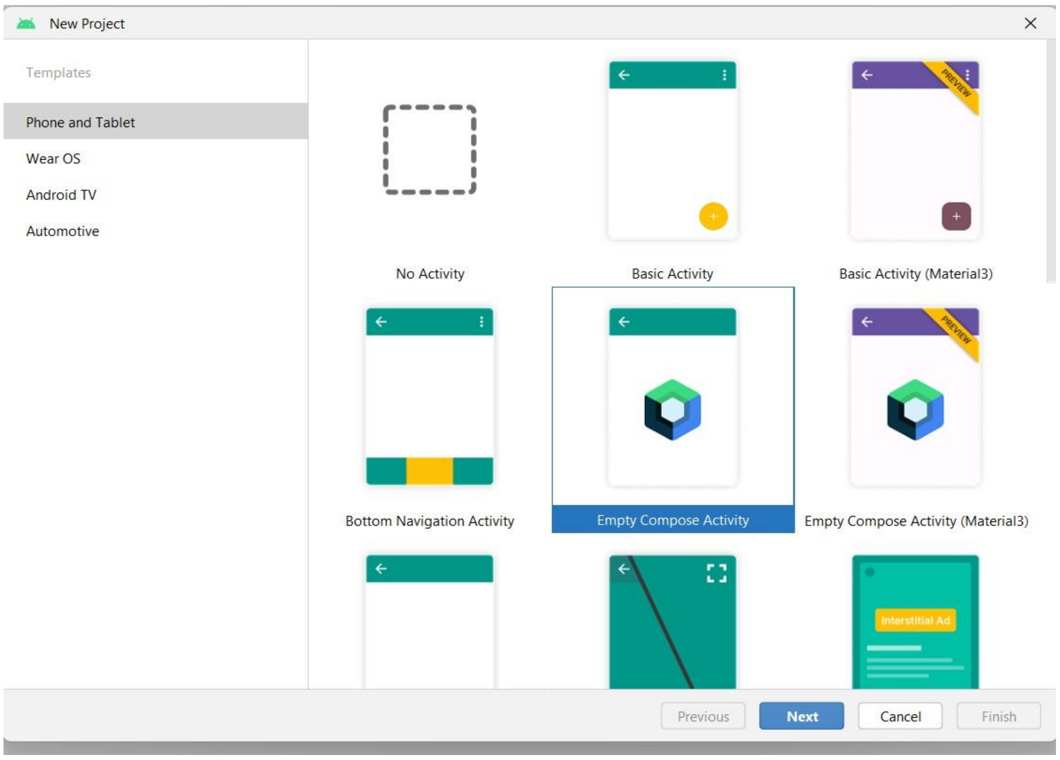
MatX

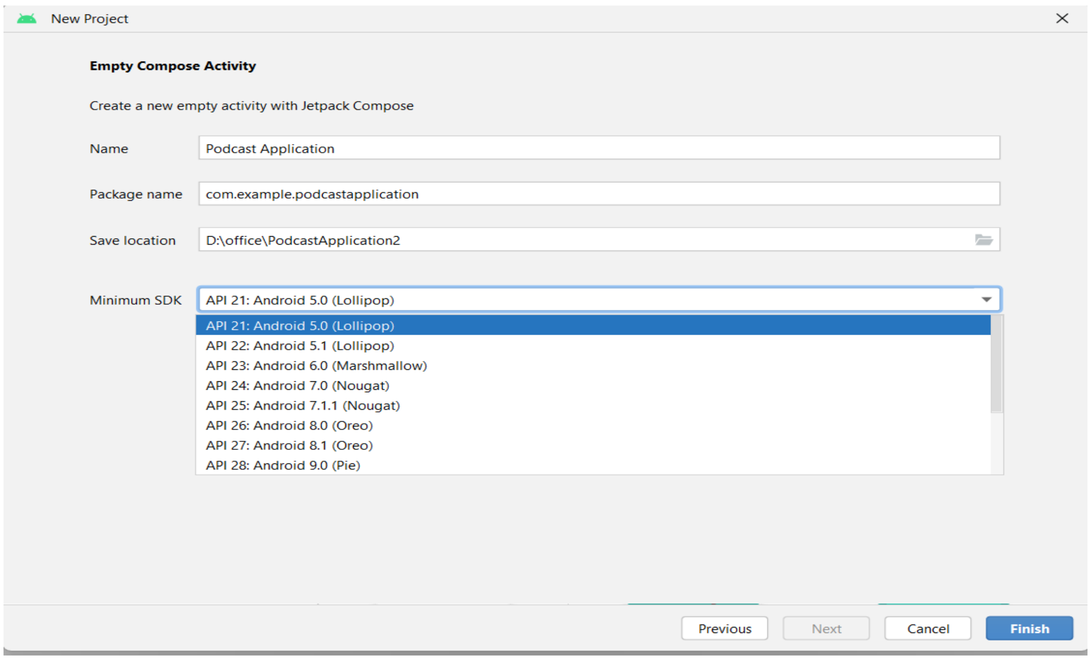
MatX is a beautifully-crafted React Native template built on top of Material Design. This Admin Dashboard template was built using React, Redux, and Material UI, and it includes all the essential features you may need to give your web application a new house. The free version of MatX can be used to easily set up admin panels, user management systems, and project management systems.

Features:

- Material UI components and elements
- Dashboard and analytics views
- Beautiful palette combination

Creating a new project




New Project

Empty Compose Activity

Create a new empty activity with Jetpack Compose

Name: Podcast Application

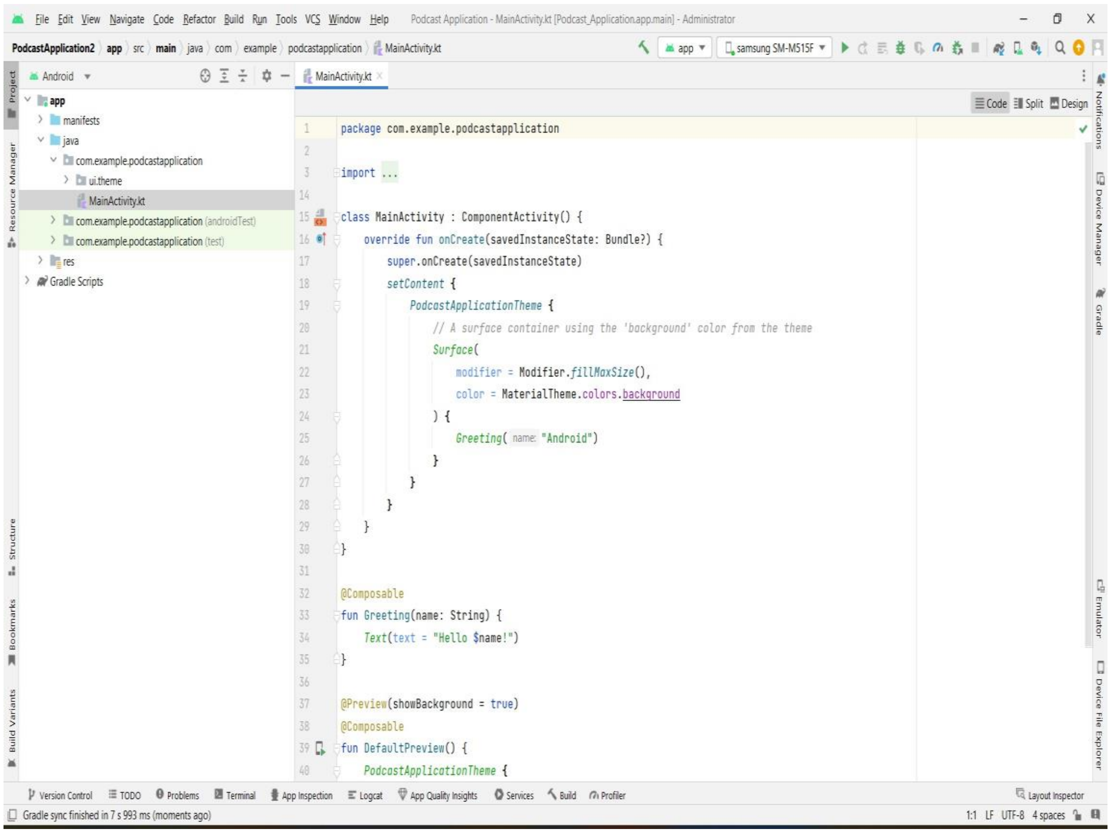
Package name: com.example.podcastapplication

Save location: D:\office\PodcastApplication2

Minimum SDK: API 21: Android 5.0 (Lollipop)

API 21: Android 5.0 (Lollipop)
API 22: Android 5.1 (Lollipop)
API 23: Android 6.0 (Marshmallow)
API 24: Android 7.0 (Nougat)
API 25: Android 7.1.1 (Nougat)
API 26: Android 8.0 (Oreo)
API 27: Android 8.1 (Oreo)
API 28: Android 9.0 (Pie)

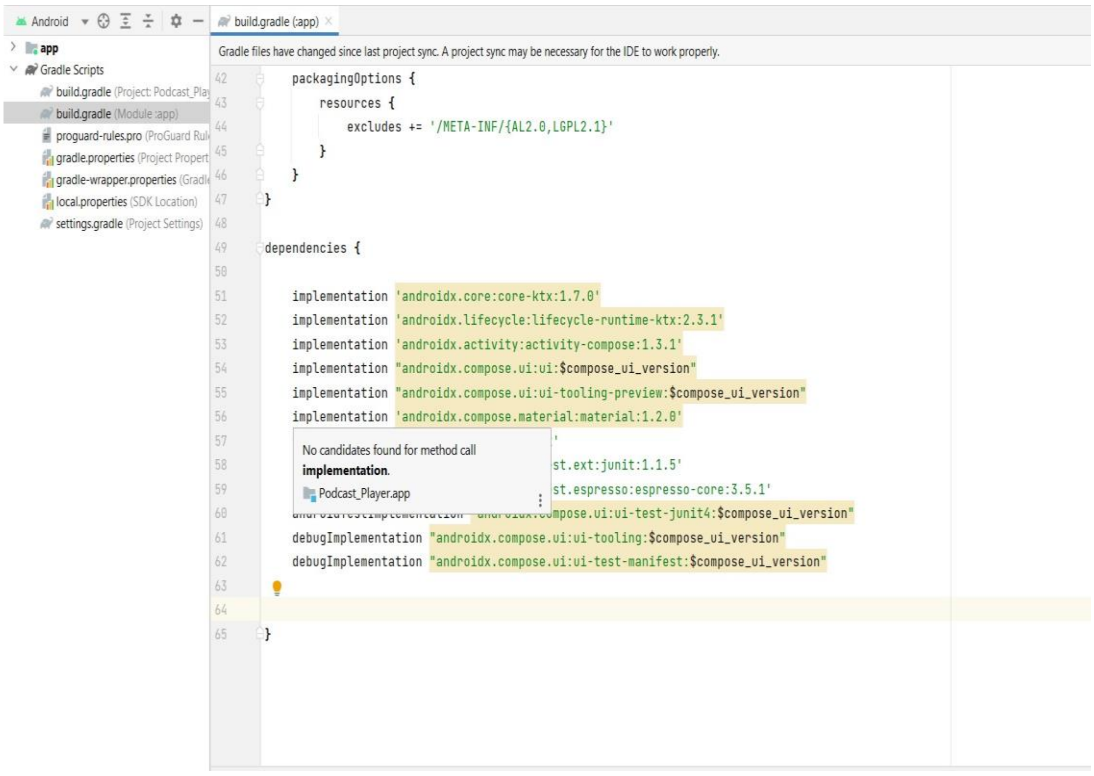
Main activity file


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help   Podcast Application - MainActivity.kt [Podcast_Application.app.main] - Administrator

PodcastApplication2 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 podcastapplication 〉 MainActivity.kt

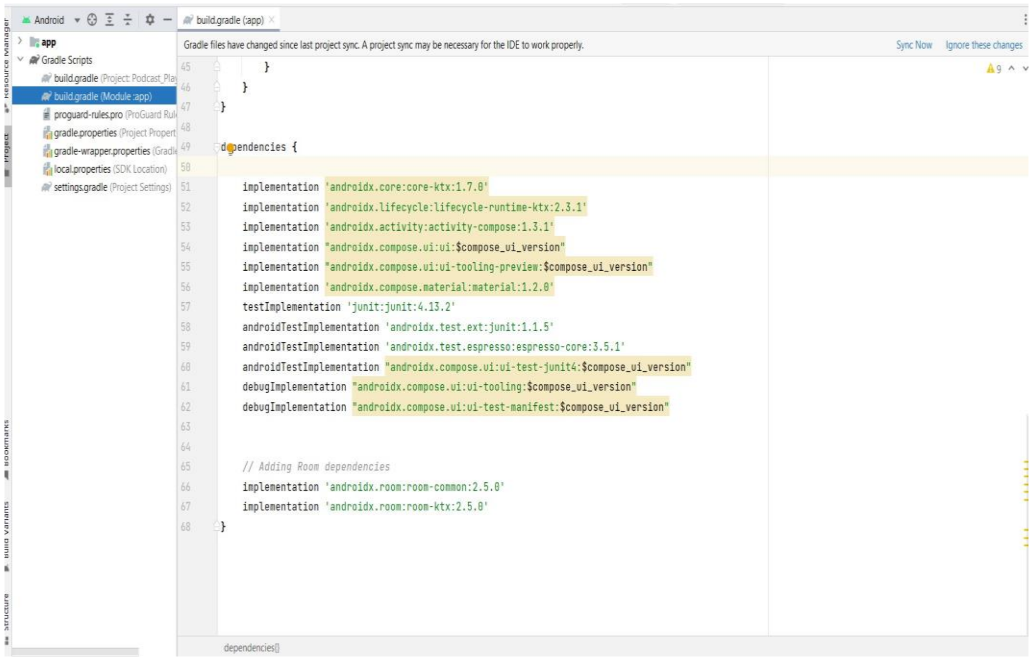
```
package com.example.podcastapplication

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            PodcastApplicationTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    Greeting( name: "Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    PodcastApplicationTheme {
```

Gradle sync finished in 7 s 993 ms (moments ago)


Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

```
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
        }
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.0'
                                             '
                                             st.ext:junit:1.1.5'
                                             st.espresso:espresso-core:3.5.1'
                                   mpose.ui:ui-test-junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"


}
```

No candidates found for method call **implementation**.
Podcast_Player.app


Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

```
        }
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-manifest:$compose_ui_version"


    // Adding Room dependencies
    implementation 'androidx.room:room-common:2.5.0'
    implementation 'androidx.room:room-ktx:2.5.0'
}
```

Android
build.gradle (:app)
Sync Now
Ignore these changes
app
Gradle Scripts
build.gradle (Project: Podcast_Play
build.gradle (Module :app)
proguard-rules.pro (ProGuard Rul
gradle.properties (Project Propert
gradle-wrapper.properties (Gradle
local.properties (SDK Location)
settings.gradle (Project Settings)
Resource Manager
Project
Bookmarks
Build Variants
Structure
dependencies{}

CODE

Themes.xml

```
<resources xmlns:tools="http://schemas.android.com/tools">
        <item name="colorPrimary">@color/orange_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <item name="colorOnSecondary">@color/black</item>
        <item name="backgroundColor">@color/black</item>
        <item name="android:statusBarColor">@android:color/transparent</item>
        <item
name="android:navigationBarColor">@android:color/transparent</item>
        <item name="android:windowLightStatusBar">false</item>
        <item name="android:windowLightNavigationBar"
tools:targetApi="o_mr1">false</item>
    </style>
</resources>
```

```
package com.fabirt.podcastapp.util
  <?xml version="1.0" encoding="utf-8"?>
      package="com.fabirt.podcastapp">

      <uses-permission android:name="android.permission.INTERNET"
  />
      <uses-permission
  android:name="android.permission.FOREGROUND_SERVICE" />

      <application
          android:name=".application.PodcastApplication"
          android:allowBackup="true"
          android:fullBackupOnly="true"
          android:icon="@mipmap/ic_launcher"
```

```
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PodcastApp">
        <activity
            android:name=".ui.MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:launchMode="singleTop"
            android:screenOrientation="portrait"
            android:theme="@style/Theme.PodcastApp.Launch"
            android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category
android:name="android.intent.category.DEFAULT" />
                <category
android:name="android.intent.category.BROWSABLE" />

                <data
                    android:host="www.listennotes.com"
                    android:pathPrefix="/e"
                    android:scheme="https" />
            </intent-filter>
        </activity>
```

```
            <service
                android:name=".data.service.MediaPlayerService"
                android:exported="false">
                <intent-filter>
                    <action
        android:name="android.media.browse.MediaBrowserService" />
                </intent-filter>
            </service>
        </application>

        </manifest>
```

```
plugins
{
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-kapt'
    id 'dagger.hilt.android.plugin'
}

def localProperties = new Properties()
localProperties.load(new FileInputStream(rootProject.file("local.properties")))

android {
    compileSdk 30
    buildToolsVersion "31.0.0"

    defaultConfig {
        applicationId "com.fabirt.podcastapp"
        minSdk 26
        targetSdk 30
        versionCode 2
        versionName "1.0.1"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }

        buildConfigField "String", "API_KEY", "\"" + localProperties['apiKey']
+ "\""
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
        useIR = true
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion compose_version
        kotlinCompilerVersion '1.4.32'
    }
```

```
}

dependencies {

    implementation 'androidx.core:core-ktx:1.5.0'
    implementation 'androidx.appcompat:appcompat:1.3.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version"
    implementation "androidx.compose.ui:ui-tooling:$compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.4.0-alpha02'
    implementation 'androidx.activity:activity-compose:1.3.0-beta02'
    implementation "androidx.lifecycle:lifecycle-viewmodel-compose:1.0.0-
alpha07"
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    androidTestImplementation "androidx.compose.ui:ui-test-
junit4:$compose_version"

    // Kotlin
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_version"
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-
core:$kotlin_coroutines_version"
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-
android:$kotlin_coroutines_version"



    // Navigation
    implementation "androidx.navigation:navigation-compose:2.4.0-alpha03"

    // Compose Accompanist
    implementation "com.google.accompanist:accompanist-
insets:$accompanist_version"
    implementation "com.google.accompanist:accompanist-
coil:$accompanist_version"

    // Hilt - dependency injection
    implementation "com.google.dagger:hilt-android:$hilt_version"
    kapt "com.google.dagger:hilt-compiler:$hilt_version"
    implementation 'androidx.hilt:hilt-lifecycle-viewmodel:1.0.0-alpha03'
    kapt 'androidx.hilt:hilt-compiler:1.0.0'

    // Retrofit
    implementation "com.squareup.retrofit2:retrofit:$retrofit_version"
    implementation "com.squareup.retrofit2:converter-gson:$retrofit_version"

    // Preferences DataStore
    implementation "androidx.datastore:datastore-preferences:1.0.0-beta02"

    // ExoPlayer
    implementation
"com.google.android.exoplayer:exoplayer:$exo_player_version"
    implementation "com.google.android.exoplayer:extension-
mediasession:$exo_player_version"

    // Glide image loading
    implementation "com.github.bumptech.glide:glide:$glide_version"

    // Palette API - Selecting colors
//    implementation 'com.android.support:palette-v7:28.0.0'
    implementation 'androidx.palette:palette-ktx:1.0.0'
}
```

PlaybackStateCompatExt.kt

```
package com.fabirt.podcastapp.util
import android.os.SystemClock
import android.support.v4.media.session.PlaybackStateCompat

inline val PlaybackStateCompat.isPrepared: Boolean
    get() = state == PlaybackStateCompat.STATE_BUFFERING ||
            state == PlaybackStateCompat.STATE_PLAYING ||
            state == PlaybackStateCompat.STATE_PAUSED

inline val PlaybackStateCompat.isPlaying: Boolean
    get() = state == PlaybackStateCompat.STATE_BUFFERING ||
            state == PlaybackStateCompat.STATE_PLAYING

inline val PlaybackStateCompat.isPlayEnabled: Boolean
    get() = actions and PlaybackStateCompat.ACTION_PLAY != 0L ||
            (actions and PlaybackStateCompat.ACTION_PLAY_PAUSE != 0L &&
                    state == PlaybackStateCompat.STATE_PAUSED)

inline val PlaybackStateCompat.isStopped: Boolean
    get() = state == PlaybackStateCompat.STATE_NONE ||
            state == PlaybackStateCompat.STATE_ERROR

inline val PlaybackStateCompat.isError: Boolean
    get() = state == PlaybackStateCompat.STATE_ERROR

inline val PlaybackStateCompat.currentPosition: Long
    get() = if (state == PlaybackStateCompat.STATE_PLAYING) {
        val timeDelta = SystemClock.elapsedRealtime() - lastPositionUpdateTime
        (position + (timeDelta * playbackSpeed)).toLong()
    } else position
```

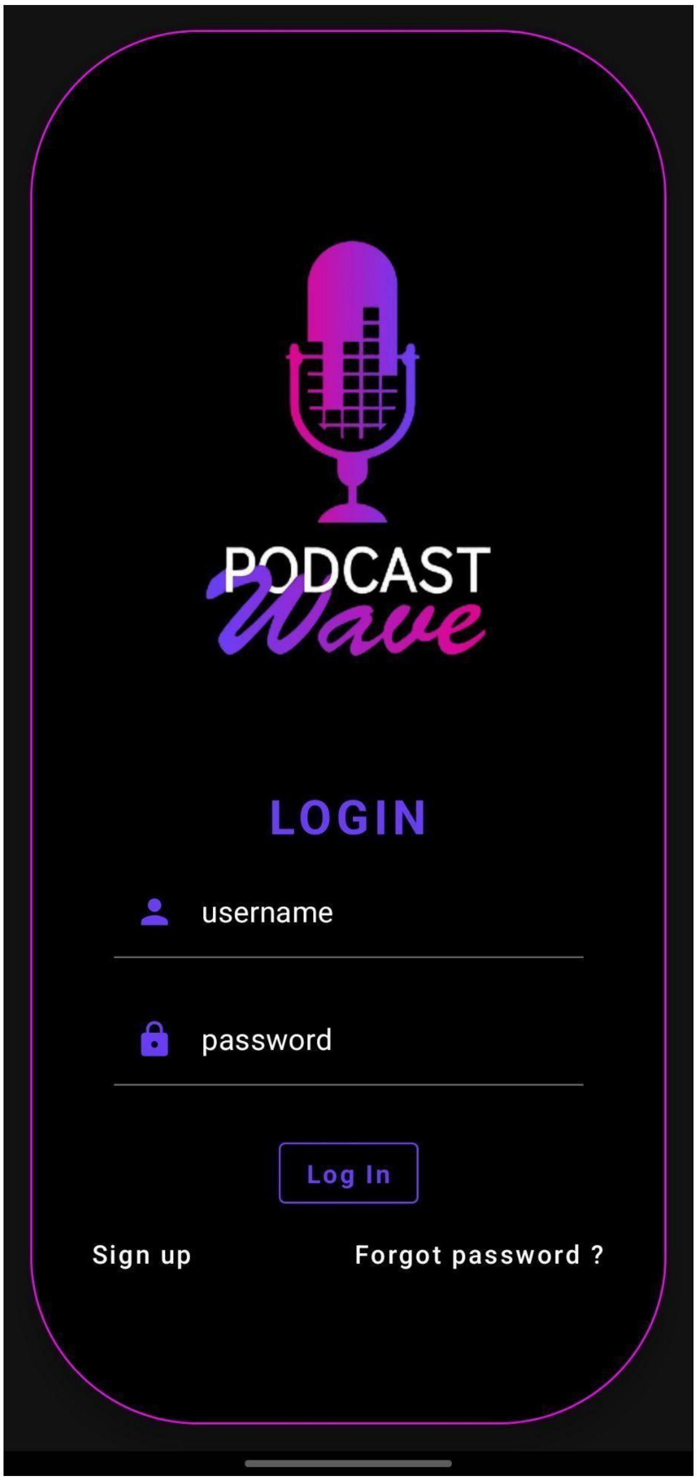
MediaPlayerService.kt

```
package com.fabirt.podcastapp.data.service
```

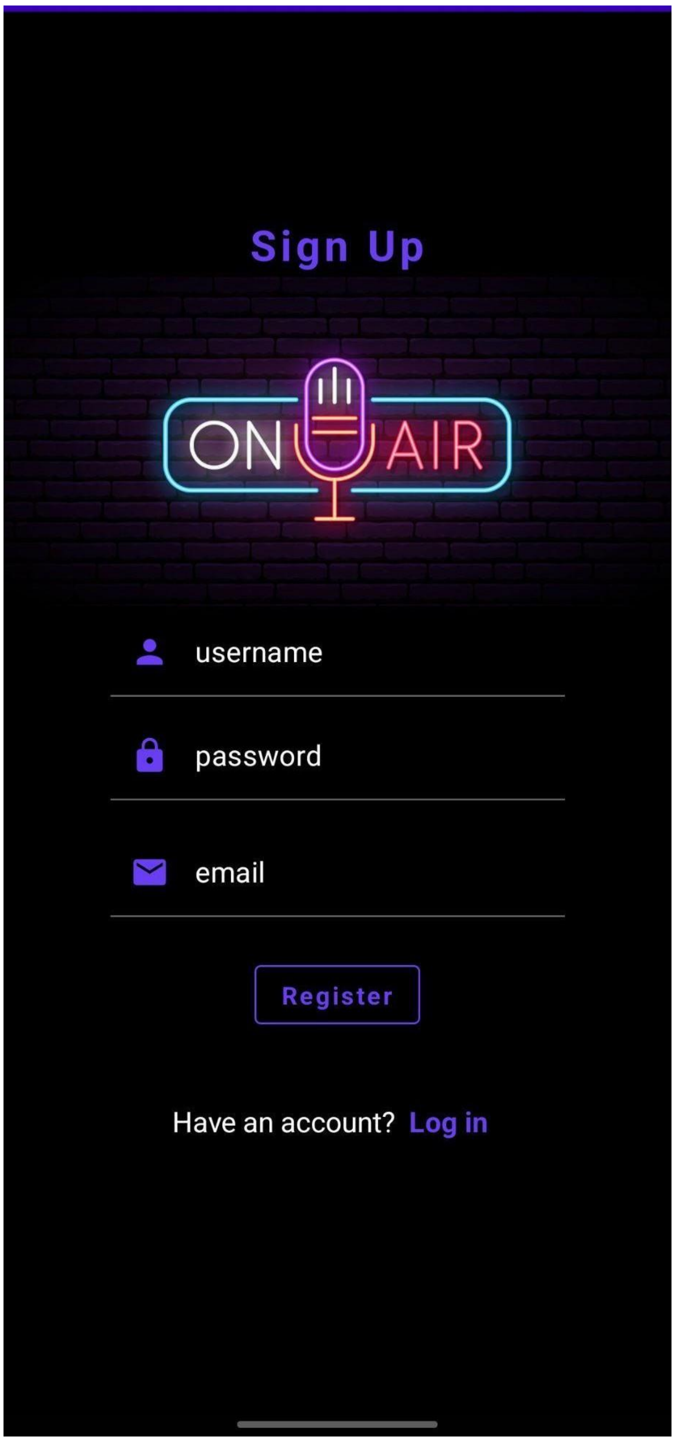
```kotlin
import android.app.Service
import android.content.Intent
import android.os.Bundle
import android.support.v4.media.MediaBrowserCompat
import android.support.v4.media.MediaMetadataCompat
import android.support.v4.media.session.MediaSessionCompat
import android.util.Log
import androidx.media.MediaBrowserServiceCompat
import com.fabirt.podcastapp.constant.K
import com.fabirt.podcastapp.data.exoplayer.*
import com.fabirt.podcastapp.ui.MainActivity
import com.google.android.exoplayer2.SimpleExoPlayer
import com.google.android.exoplayer2.ext.mediasession.MediaSessionConnector
import com.google.android.exoplayer2.upstream.cache.CacheDataSource
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.cancel
import javax.inject.Inject

@AndroidEntryPoint
class MediaPlayerService : MediaBrowserServiceCompat() {
    @Inject
    lateinit var dataSourceFactory: CacheDataSource.Factory

    @Inject
    lateinit var exoPlayer: SimpleExoPlayer

    @Inject
    lateinit var mediaSource: PodcastMediaSource

    private val serviceJob = Job()
    private val serviceScope = CoroutineScope(Dispatchers.Main + serviceJob)

    private lateinit var mediaSession: MediaSessionCompat
    private lateinit var mediaSessionConnector: MediaSessionConnector

    private lateinit var mediaPlayerNotificationManager:
MediaPlayerNotificationManager

    private var currentPlayingMedia: MediaMetadataCompat? = null

    private var isPlayerInitialized = false

    var isForegroundService: Boolean = false

    companion object {
        private const val TAG = "MediaPlayerService"

        var currentDuration: Long = 0L
            private set
    }

    override fun onCreate() {
        super.onCreate()
        Log.i(TAG, "onCreate called")
        val activityPendingIntent = Intent(this, MainActivity::class.java)
            .apply {
                action = K.ACTION_PODCAST_NOTIFICATION_CLICK
            }
            .let {
                PendingIntent.getActivity(
                    this,
                    0,
                    it,
                    PendingIntent.FLAG_UPDATE_CURRENT or
PendingIntent.FLAG_IMMUTABLE
                )
            }

        mediaSession = MediaSessionCompat(this, TAG).apply {
            setSessionActivity(activityPendingIntent)
            isActive = true
        }

        val mediaPlaybackPreparer = MediaPlaybackPreparer(mediaSource) {
mediaMetadata ->
            currentPlayingMedia = mediaMetadata
            preparePlayer(mediaSource.mediaMetadataEpisodes, mediaMetadata,
true)
        }
        mediaSessionConnector = MediaSessionConnector(mediaSession).apply {
            setPlaybackPreparer(mediaPlaybackPreparer)
            setQueueNavigator(MediaPlayerQueueNavigator(mediaSession,
mediaSource))
            setPlayer(exoPlayer)
        }

        this.sessionToken = mediaSession.sessionToken

        mediaPlayerNotificationManager = MediaPlayerNotificationManager(
            this,
            mediaSession.sessionToken,
            MediaPlayerNotificationListener(this)
        ) {
            currentDuration = exoPlayer.duration
        }
    }
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        return Service.START_STICKY
    }

    override fun onCustomAction(action: String, extras: Bundle?, result:
Result<Bundle>) {
        super.onCustomAction(action, extras, result)
        when (action) {
            K.START_MEDIA_PLAYBACK_ACTION -> {
                mediaPlayerNotificationManager.showNotification(exoPlayer)
            }
            K.REFRESH_MEDIA_BROWSER_CHILDREN -> {
                mediaSource.refresh()
                notifyChildrenChanged(K.MEDIA_ROOT_ID)
            }
            else -> Unit
        }
    }

    override fun onGetRoot(
        clientPackageName: String,
        clientUid: Int,
        rootHints: Bundle?
    ): BrowserRoot {
        return BrowserRoot(K.MEDIA_ROOT_ID, null)
    }

    override fun onLoadChildren(
        parentId: String,
        result: Result<MutableList<MediaBrowserCompat.MediaItem>>
    ) {
        Log.i(TAG, "onLoadChildren called")
        when (parentId) {
            K.MEDIA_ROOT_ID -> {
                val resultsSent = mediaSource.whenReady { isInitialized ->
                    if (isInitialized) {

                        result.sendResult(mediaSource.asMediaItems())
                        if (!isPlayerInitialized &&
mediaSource.mediaMetadataEpisodes.isNotEmpty()) {
                            isPlayerInitialized = true
                        }
                    } else {
                        result.sendResult(null)
                    }
                }
                if (!resultsSent) {
                    result.detach()
                }
            }
            else -> Unit
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        serviceScope.cancel()
        exoPlayer.release()
    }

    private fun preparePlayer(
        mediaMetaData: List<MediaMetadataCompat>,
        itemToPlay: MediaMetadataCompat?,
        playWhenReady: Boolean
    ) {
        val indexToPlay = if (currentPlayingMedia == null) 0 else
mediaMetaData.indexOf(itemToPlay)

exoPlayer.setMediaSource(mediaSource.asMediaSource(dataSourceFactory))
        exoPlayer.prepare()
        exoPlayer.seekTo(indexToPlay, 0L)
        exoPlayer.playWhenReady = playWhenReady
    }
}
```

OUTPUT

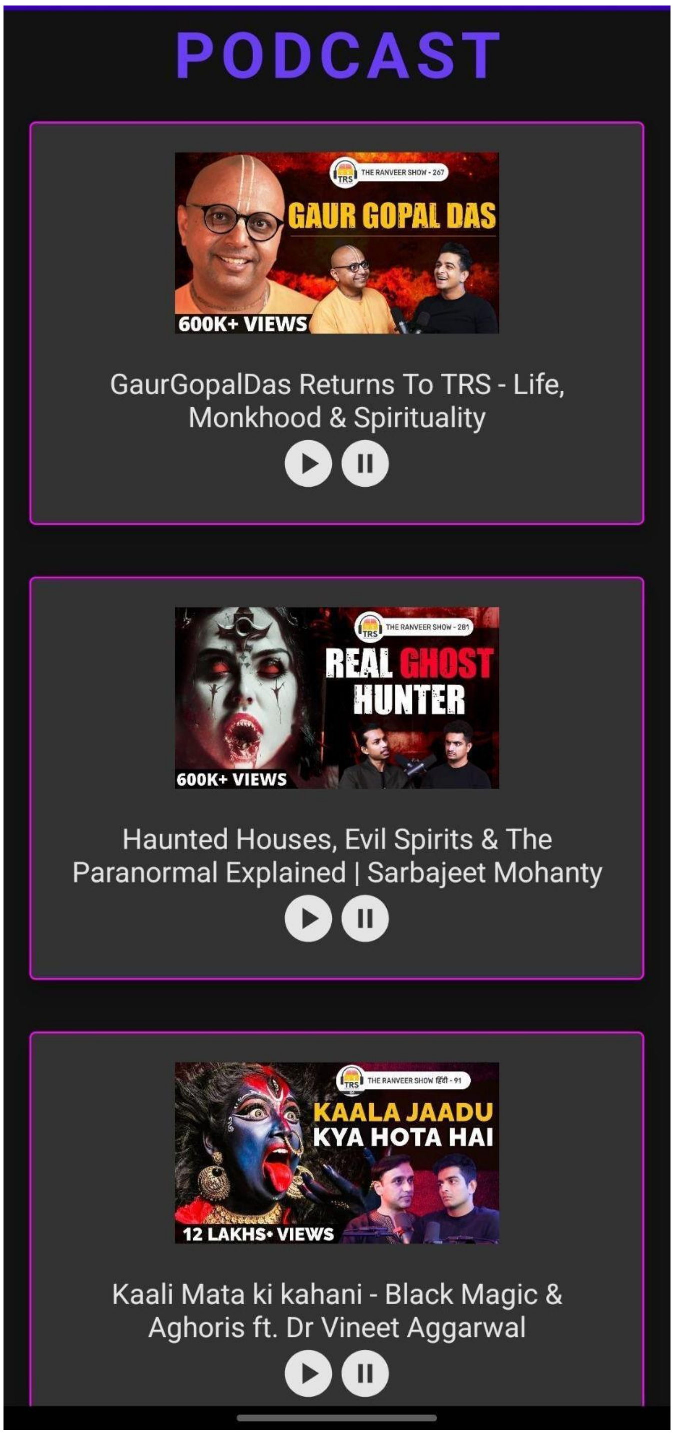
Login Page :



RegisterPage :



MainPage :



PODCAST

GaurGopalDas Returns To TRS - Life, Monkhood & Spirituality

Haunted Houses, Evil Spirits & The Paranormal Explained | Sarbajeet Mohanty

Kaali Mata ki kahani - Black Magic & Aghoris ft. Dr Vineet Aggarwal

CONCLUSIONS

Today's Internet user expects to experience personalized interaction with websites. If the company fails to deliver they run the risk of losing a potential customer forever. An important aspect of creating interactive web forms to collect information from users is to be able to check that the information entered is valid, therefore; information submitted through these forms should be extensively validated. Validation could be performed using client script where errors are detected when the form is submitted to the server and if any errors are found the submission of the form to the server is cancelled and all errors displayed to the user. This allows the user to correct their input before re-submitting the form to the server. We can not underestimate the importance of input validation which ensures that the application is robust against all forms of input data obtained from the user.