

Podcast Plus: A Redux-Inspired Podcast App

With Dynamic Themes For Android

1. INTRODUCTION

1.1 OVERVIEW:

A podcast is a program made available in digital format for download over the Internet. an episodic series of digital audio files that a user can download to a personal device to listen to at a time of their choosing. Podcasts are primarily an audio medium, with some programs offering a supplemental video component. Streaming applications and podcasting services provide a convenient and integrated way to manage a personal consumption queue across many podcast sources and playback devices. There are also podcast search engines, which help users find and share podcast episodes.

Podcast episodes are widely stored and encoded in the mp3 digital audio format and then hosted on dedicated or shared webserver space. Syndication of podcasts' episodes across various websites and platforms is based on RSS feeds, an XML-formatted file citing information about the episode and the podcast itself.

The most basic equipment for a podcast is a computer and a microphone. It is helpful to have a sound-proof room and headphones. The computer should have a recording or streaming application installed. Typical microphones for podcasting are connected using USB. If the podcast involves two or more people, each person requires a microphone, and a USB audio interface is needed to mix them together. If the podcast includes video (livestreaming), then a separate webcam might be needed, and additional lighting.

1.2 PURPOSE:

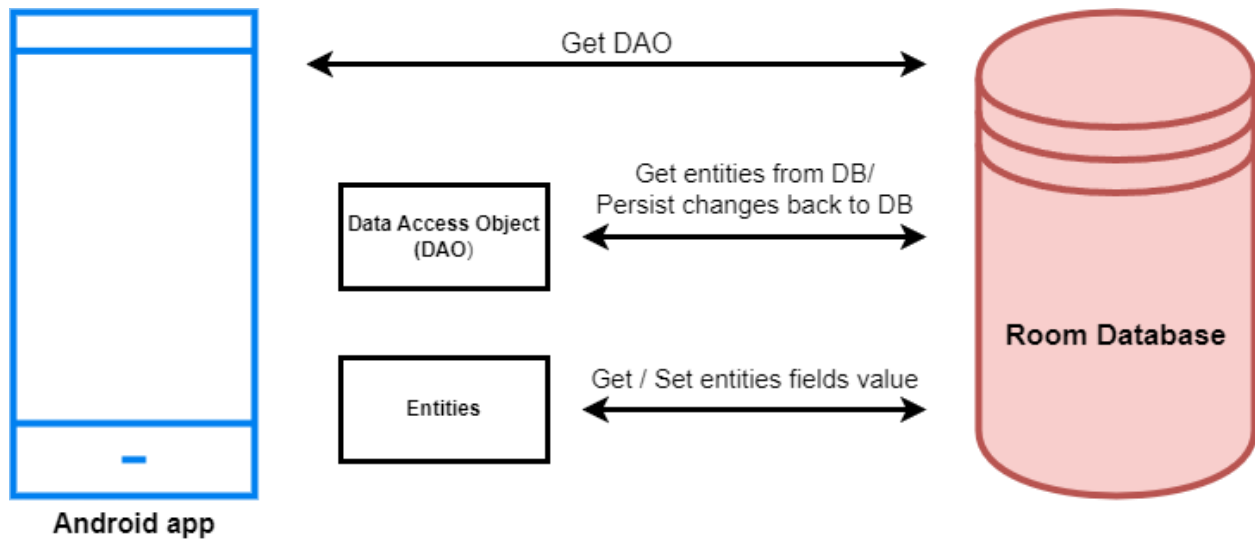
Podcast Plus is a premium subscription service that offers additional features and benefits to users of a podcasting platform or app. The purpose of Podcast Plus can vary depending on the specific platform or app, but generally, it serves the following purposes:

Enhanced Listening Experience: Podcast Plus may provide users with an enhanced listening experience by offering features such as ad-free listening, higher audio quality, early access to episodes, and the ability to download episodes for offline listening. This allows users to enjoy their favorite podcasts without interruptions, with improved audio clarity, and the convenience of listening on-the-go even without an internet connection.

Exclusive Content: Podcast Plus may offer subscribers exclusive content that is not available to free users. This could include bonus episodes, behind-the-scenes content, extended interviews, or special episodes that are only accessible to subscribers. This exclusive content gives users added value for their subscription and encourages them to support their favorite podcasters by subscribing to Podcast Plus.

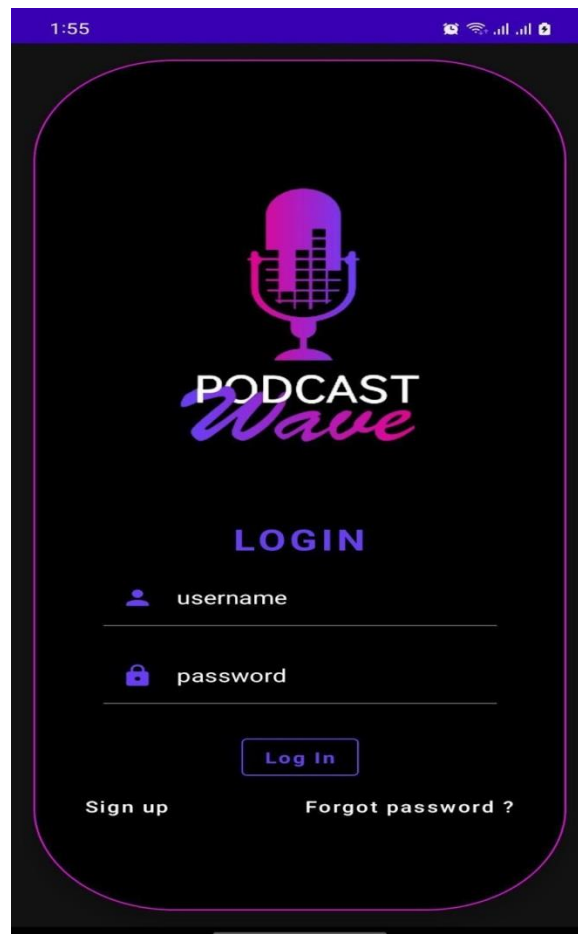
2. PROBLEM DEFINITION AND DESIGN THINKING:

2.1 EMPATHY MAP:

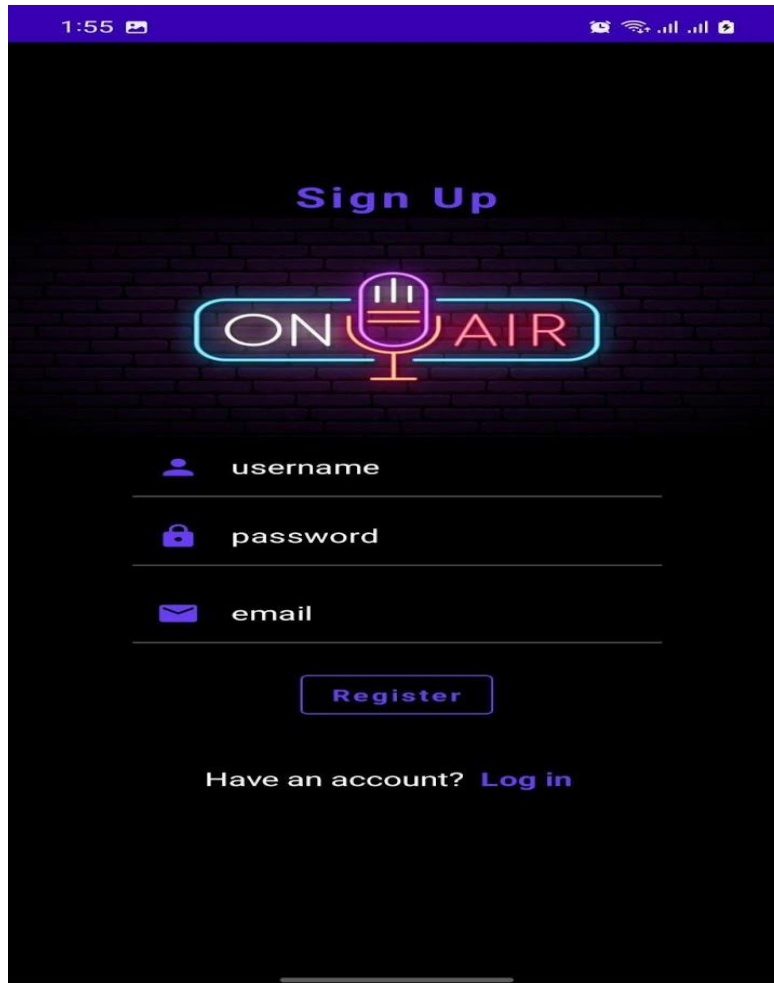


3. RESULT:

Login Page:



Register Page:



1:55

Sign Up

ON AIR

username

password

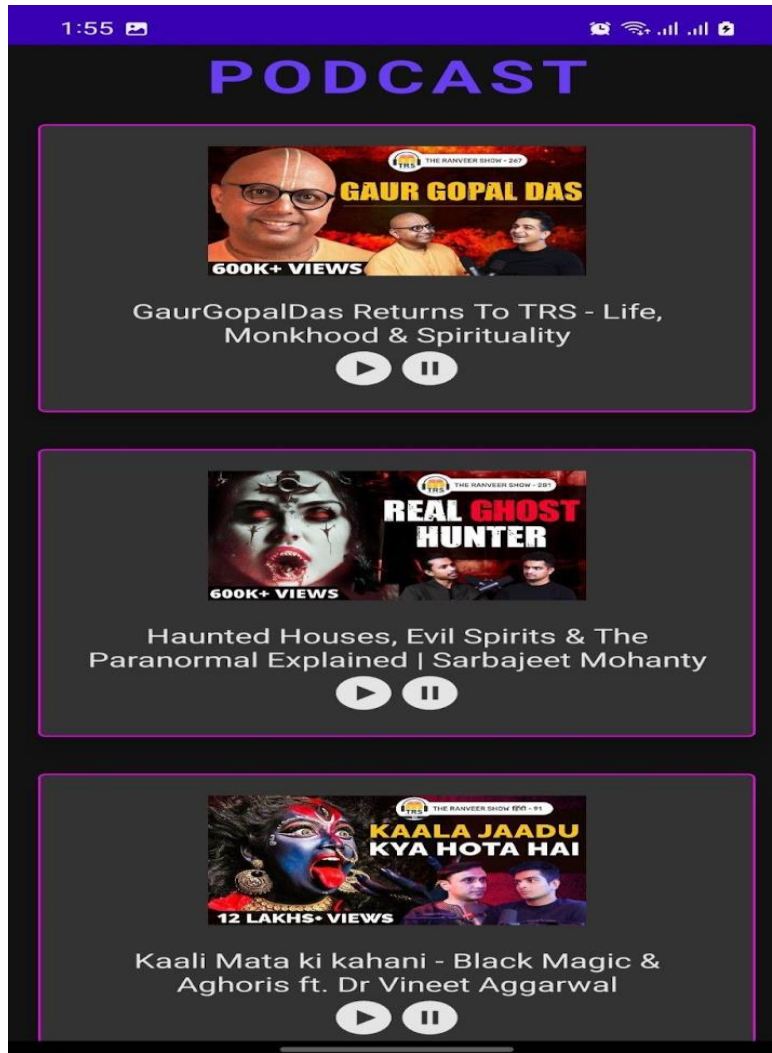
email

Register

Have an account? [Log in](#)

The image shows a mobile application registration screen. At the top, there is a status bar with the time 1:55 and various icons. The main heading is 'Sign Up' in a light blue font. Below this is a logo for 'ON AIR' where the 'O' and 'A' are in light blue and the 'N' and 'I' are in pink, with a microphone icon in the center. The background is a dark brick wall. There are three input fields: 'username' with a person icon, 'password' with a lock icon, and 'email' with an envelope icon. A light blue 'Register' button is centered below the fields. At the bottom, it says 'Have an account? [Log in](#)'.

Main Podcast Page:



4. ADVANTAGES AND DISADVANTAGES

Advantages:

Podcast Plus follows the Redux design pattern, which is a popular architecture for building scalable and maintainable mobile apps. This design pattern helps ensure that the app's state management is well-organized, efficient, and predictable, resulting in a smooth and reliable user experience.

Podcast Plus offers dynamic themes that allow users to customize the app's appearance according to their preferences. This feature enables users to personalize the app's look and feel, making it more visually appealing and enjoyable to use.

Podcast Plus is designed to provide a seamless and user-friendly experience for podcast listeners. It offers features such as easy podcast discovery, intuitive episode navigation, and customizable playlists, making it convenient for users to find, listen to, and organize their favorite podcasts.

Podcast Plus provides access to a diverse range of podcasts covering various genres, topics, and languages. This allows users to explore and discover new podcasts that cater to their interests, making the app suitable for all types of podcast enthusiasts.

Podcast Plus allows users to download episodes for offline listening. This is especially useful when users are traveling or in areas with limited internet connectivity, ensuring uninterrupted podcast listening even without an internet connection.

Podcast Plus is actively maintained and updated, with a dedicated team providing regular bug fixes, feature enhancements, and customer support. This ensures that users have a reliable and up-to-date podcast app with a responsive support system.

Disadvantages:

While Redux-inspired and dynamic themes may appeal to certain tech-savvy and design-conscious users, it may not necessarily cater to a wider audience. The app may have a smaller user base compared to more mainstream podcast apps that offer a more traditional user interface.

Redux is a state management library that requires a solid understanding of concepts like actions, reducers, and stores. Users who are not familiar with Redux may find it challenging to navigate and fully utilize the features of the app. This could potentially limit the adoption of the app among less technically inclined users.

Implementing dynamic themes in an app can introduce complexity in terms of design, development, and maintenance. Managing different themes dynamically may require additional resources and efforts, including regular updates to support new themes, and could increase the risk of potential bugs or inconsistencies in the user interface.

While dynamic themes may offer flexibility in terms of changing the look and feel of the app, they may have limitations in terms of customization options. Users may have specific preferences for color schemes, fonts, and other visual elements that may not be fully supported by the dynamic themes offered by the app, leading to potential dissatisfaction among users.

As Podcast Plus is specifically designed for Android, it may not be available for users who prefer other platforms such as iOS or web-based podcast apps. This could limit the app's reach to a broader user base.

The podcast app market is highly competitive, with several established players and a plethora of alternative options available to users. Differentiating and gaining market share may be challenging for a newer app like Podcast Plus, even with unique features like Redux-inspired architecture and dynamic themes.

5. APPLICATION

Podcast Plus is a podcast application for Android that is inspired by the Redux design pattern and features dynamic themes. Here are some potential uses for the Podcast Plus application:

1. **Discover and Listen to Podcasts:** Podcast Plus allows users to discover and listen to their favorite podcasts. Users can search for podcasts based on their interests, browse popular podcasts, and subscribe to their preferred podcasts for easy access to new episodes. They can also listen to podcasts on demand or download episodes for offline listening.
2. **Dynamic Themes:** One of the unique features of Podcast Plus is its dynamic themes. Users can customize the look and feel of the application by choosing from different themes that are dynamically applied in real-time. This allows users to personalize their podcast listening experience and change the theme to match their mood or preferences.
3. **Podcast Management:** Podcast Plus provides tools for users to manage their podcast subscriptions ,including organizing podcasts into playlists, marking episodes as played or unplayed, and setting up automatic episode downloads. Users can also customize the playback speed, skip intervals, and sleep timer for a tailored listening experience.
4. **Offline Listening:** Podcast Plus allows users to download episodes for offline listening, making it convenient for users who are on the go or in areas with limited internet connectivity. Users can download episodes in advance and listen to them later without needing an internet connection, perfect for long flights, road trips, or other situations where internet access may be limited.
5. **Playlist Creation:** Podcast Plus enables users to create playlists of their favorite episodes or podcasts. Users can curate their own personalized playlists based on their interests, mood, or listening preferences. Playlists can be created for different occasions or purposes, such as creating a playlist for a workout session, a relaxing playlist for bedtime, or a playlist for commuting.

6.CONCLUSION

In conclusion, the Podcast Plus application is a cutting-edge podcast app for Android users that offers a unique and innovative feature set. Inspired by Redux, a popular JavaScript library for managing application state, Podcast Plus leverages dynamic themes to provide a customizable and visually appealing user experience.

With its intuitive interface and powerful functionalities, Podcast Plus is poised to become a go-to app for podcast enthusiasts.

One of the standout features of Podcast Plus is its dynamic themes, which allow users to personalize the app's appearance according to their preferences. This feature provides a fresh and engaging experience, making the app visually appealing and enjoyable to use.

7.FUTURE SCOPE

The Future scopes for a Redux-Inspired Podcast App with Dynamic Themes for Android, which we will refer to as "Podcast Plus" application.

1. **Customized User Profiles:** Implementing user profiles that allow listeners to create personalized accounts within the app. This could include features such as a history of previously played episodes, saved episodes, and playlists. Users could also have the ability to sync their profiles across multiple devices for a seamless listening experience.
2. **Social Media Integration:** Adding social media integration to allow users to share their favorite episodes or playlists on social media platforms such as Facebook, Twitter, and Instagram.
3. **Discover and Recommendation Features:** Enhancing the app's recommendation algorithm to provide users with personalized episode recommendations based on their listening history, interests, and preferences..
4. **Enhanced Search and Filtering Options:** Improving the search functionality within the app, allowing users to search for episodes by keywords, tags, or categories.
5. **Interactive Community Features:** Creating a community within the app where users can interact with each other, share recommendations, and discuss their favorite episodes.
6. **Monetization Opportunities:** Exploring different monetization options such as in-app purchases, premium subscriptions, and sponsored content to generate revenue for the app. This could include offering ad-free listening, exclusive content, or premium features to paid subscribers.
7. **Accessibility Features:** Incorporating accessibility features such as closed captioning, transcripts, and adjustable playback speed to make the app more inclusive and accessible to users with different needs and preferences.

8.APPENDIX

A.Source Code:

LOGINACTIVITY.KT

```
package com.example.podcastapplication

import android.os.Bundle

import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview

import com.example.podcastapplication.ui.theme.PodcastApplicationTheme

class loginactivity : 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  
) {  
    Greeting2("Android")  
}  
}  
}  
}  
}
```

@Composable

```
fun Greeting2(name: String) {  
    Text(text = "Hello $name!")  
}
```

@Preview(showBackground = true)

@Composable

```
fun DefaultPreview2() {  
    PodcastApplicationTheme {  
        Greeting2("Android")  
    }  
}
```

BUILD.GRADLE

```
plugins {  
    id 'com.android.application'  
    id 'org.jetbrains.kotlin.android'  
}
```

```
android {  
    namespace 'com.example.podcastapplication'  
    compileSdk 33  
  
    defaultConfig {  
        applicationId "com.example.podcastapplication"  
        minSdk 21  
        targetSdk 33  
        versionCode 1  
        versionName "1.0"  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
        vectorDrawables {  
            useSupportLibrary true  
        }  
    }  
  
    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'  
}  
  
buildFeatures {  
    compose true  
}  
  
composeOptions {  
    kotlinCompilerExtensionVersion '1.2.0'  
}  
  
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'  
    testImplementation 'junit:junit:4.13.2'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
```

```
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
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 '
}
```

USER.KT

```
package com.example.podcastapplication

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "user_table")

data class User(

    @PrimaryKey(autoGenerate = true) val id: Int?,

    @ColumnInfo(name = "first_name") val firstName: String?,

    @ColumnInfo(name = "last_name") val lastName: String?,

    @ColumnInfo(name = "email") val email: String?,

    @ColumnInfo(name = "password") val password: String?,

)
```

USERDAO.KT

```
package com.example.podcastapplication

import androidx.room.*
```



```

@Dao

interface UserDao {

    @Query("SELECT * FROM user_table WHERE email = :email")

    suspend fun getUserByEmail(email: String): User?

    @Insert(onConflict = OnConflictStrategy.REPLACE)

    suspend fun insertUser(user: User)

    @Update

    suspend fun updateUser(user: User)

    @Delete

    suspend fun deleteUser(user: User)

}

```

USERDATABASE.KT

```

package com.example.podcastapplication

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [User::class], version = 1)

abstract class UserDatabase : RoomDatabase() {

    abstract fun userDao(): UserDao

    companion object {

        @Volatile

        private var instance: UserDatabase? = null

        fun getDatabase(context: Context): UserDatabase {

            return instance ?: synchronized(this) {

```

```

val newInstance = Room.databaseBuilder(
    context.applicationContext,
    UserDatabase::class.java,
    "user_database"
).build()

instance = newInstance

newInstance
}
}
}
}

```

USERDATABASEHELPER.KT

```

package com.example.podcastapplication

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    companion object {

        private const val DATABASE_VERSION = 1

        private const val DATABASE_NAME = "UserDatabase.db"

        private const val TABLE_NAME = "user_table"
    }
}

```

```

private const val COLUMN_ID = "id"

private const val COLUMN_FIRST_NAME = "first_name"

private const val COLUMN_LAST_NAME = "last_name"

private const val COLUMN_EMAIL = "email"

private const val COLUMN_PASSWORD = "password"

}

override fun onCreate(db: SQLiteDatabase?) {

val createTable = "CREATE TABLE $TABLE_NAME (" +

"$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +

"$COLUMN_FIRST_NAME TEXT, " +

"$COLUMN_LAST_NAME TEXT, " +

"$COLUMN_EMAIL TEXT, " +

"$COLUMN_PASSWORD TEXT" +

")"

db?.execSQL(createTable)

}

override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {

db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")

onCreate(db)

}


fun insertUser(user: User) {

val db = writableDatabase

val values = ContentValues()

values.put(COLUMN_FIRST_NAME, user.firstName)

```

```

values.put(COLUMN_LAST_NAME, user.lastName)

values.put(COLUMN_EMAIL, user.email)

values.put(COLUMN_PASSWORD, user.password)

db.insert(TABLE_NAME, null, values)

db.close()

}

```

```

@SuppressLint("Range")

fun getUserByUsername(username: String): User? {

    val db = readableDatabase

    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
    $COLUMN_FIRST_NAME = ?", arrayOf(username))

    var user: User? = null

    if (cursor.moveToFirst()) {

        user = User(

            id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

            firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),

            lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),

            email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),

            password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),

        )

    }

    cursor.close()

    db.close()

    return user
}

```

```

}

@SuppressLint("Range")

fun getUserById(id: Int): User? {

    val db = readableDatabase

    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
    $COLUMN_ID = ?", arrayOf(id.toString()))

    var user: User? = null

    if (cursor.moveToFirst()) {

        user = User(

            id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),

            firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),

            lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),

            email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),

            password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),

        )

    }

    cursor.close()

    db.close()

    return user

}

@SuppressLint("Range")

fun getAllUsers(): List<User> {

    val users = mutableListOf<User>()

    val db = readableDatabase

    val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)

```

```

if (cursor.moveToFirst()) {
    do {
        val user = User(
            id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
            firstName = cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
            lastName = cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
            email = cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
            password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
        )
        users.add(user)
    } while (cursor.moveToNext())
}
cursor.close()
db.close()
return users
}
}

```

MAINACTIVITY.KT

```

package com.example.podcastapplication

import android.os.Bundle

import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface

```

```

import androidx.compose.material.Text

import androidx.compose.runtime.Composable

import androidx.compose.ui.Modifier

import androidx.compose.ui.tooling.preview.Preview

import com.example.podcastapplication.ui.theme.PodcastApplicationTheme

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("Android")

                }

            }

        }

    }

}

@Composable

fun Greeting(name: String) {

    Text(text = "Hello $name!")

}

@Preview(showBackground = true)

@Composable

fun DefaultPreview() {

```

```
PodcastApplicationTheme {  
    Greeting("Android")  
}  
}
```

LOGINACTIVITY.KT

```
package com.example.podcastapplication  
  
import android.os.Bundle  
  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
  
import androidx.compose.foundation.layout.fillMaxSize  
  
import androidx.compose.material.MaterialTheme  
  
import androidx.compose.material.Surface  
  
import androidx.compose.material.Text  
  
import androidx.compose.runtime.Composable  
  
import androidx.compose.ui.Modifier  
  
import androidx.compose.ui.tooling.preview.Preview  
  
import com.example.podcastapplication.ui.theme.PodcastApplicationTheme  
  
class loginactivity : 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
) {
    Greeting2("Android")
}
}
}
}
}
}

@Composable
fun Greeting2(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview2() {
    PodcastApplicationTheme {
        Greeting2("Android")
    }
}

package com.example.podcastplayerpackage com.example.podcastplayer

import android.content.Context

import android.content.Intent

import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

```

```
import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Lock
import androidx.compose.material.icons.filled.Person
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.em
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat
import com.example.podcastplayer.ui.theme.PodcastPlayerTheme
```

Class LoginActivity :

```
ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    databaseHelper = UserDatabaseHelper(this)

    setContent {
        PodcastPlayerTheme {
            // A surface container using the 'background' color from the theme

            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colors.background
            ) {
                LoginScreen(this, databaseHelper)
            }
        }
    }
}

```

@Composable

```

fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Card(
        elevation = 12.dp,
        border = BorderStroke(1.dp, Color.Magenta),

```

```
shape = RoundedCornerShape(100.dp),
modifier = Modifier.padding(16.dp).fillMaxWidth()
)
{
Column(
Modifier
.background(Color.Black)
.fillMaxHeight()
.fillMaxWidth()
.padding(bottom = 28.dp, start = 28.dp, end = 28.dp),
horizontalAlignment = Alignment.CenterHorizontally,
verticalArrangement = Arrangement.Center
)
{
Image(
painter = painterResource(R.drawable.podcast_login),
contentDescription = "", Modifier.height(400.dp).fillMaxWidth()
)
Text(
text = "LOGIN",
color = Color(0xFF6a3ef9),
fontWeight = FontWeight.Bold,
fontSize = 26.sp,
style = MaterialTheme.typography.h1,
letterSpacing = 0.1.em
```

```
)  
  
Spacer(modifier = Modifier.height(10.dp))  
  
TextField(  
    value = username,  
    onValueChange = { username = it },  
    leadingIcon = {  
        Icon(  
            imageVector = Icons.Default.Person,  
            contentDescription = "personIcon",  
            tint = Color(0xFF6a3ef9)  
        )  
    },  
    placeholder = {  
        Text(  
            text = "username",  
            color = Color.White  
        )  
    },  
    colors = TextFieldDefaults.textFieldColors(  
        backgroundColor = Color.Transparent  
    )  
)  
  
Spacer(modifier = Modifier.height(20.dp))  
  
TextField(  
    value = password,
```

```

onValueChange = { password = it },
leadingIcon = {
    Icon(
        imageVector = Icons.Default.Lock,
        contentDescription = "lockIcon",
        tint = Color(0xFF6a3ef9)
    )
},
placeholder = { Text(text = "password", color = Color.White) },
visualTransformation = PasswordVisualTransformation(),
colors = TextFieldDefaults.textFieldColors(backgroundColor = Color.Transparent)
)
Spacer(modifier = Modifier.height(12.dp))

if (error.isNotEmpty()) {
    Text(
        text = error,
        color = MaterialTheme.colors.error,
        modifier = Modifier.padding(vertical = 16.dp)
    )
}

Button(
    onClick = {
        if (username.isNotEmpty() && password.isNotEmpty()) {

```

```

val user = databaseHelper.getUserByUsername(username)

if (user != null && user.password == password) {

    error = "Successfully log in"

    context.startActivity(

        Intent(

            context,

            MainActivity::class.java

        )

    )

    //onLoginSuccess()

} else {

    error = "Invalid username or password"

}

} else {

    error = "Please fill all fields"

}

},

border = BorderStroke(1.dp, Color(0xFF6a3ef9)),

colors = ButtonDefaults.buttonColors(backgroundColor = Color.Black),

modifier = Modifier.padding(top = 16.dp)

) {

    Text(text = "Log In", fontWeight = FontWeight.Bold, color = Color(0xFF6a3ef9))

}

Row(modifier = Modifier.fillMaxWidth()) {

```

```
    TextButton(onClick = {
        context.startActivity(
            Intent(
                context,
                RegistrationActivity::class.java
            )))
    {
        Text(
            text = "Sign up",
            color = Color.White
        )
    }
    Spacer(modifier = Modifier.width(80.dp))
    TextButton(onClick = { /* Do something! */ })
    {
        Text(
            text = "Forgot password ?",
            color = Color.White
        )
    }
}
}
}

fun startMainPage(context: Context) {
    val intent = Intent(context, MainActivity::class.java)
```



```
ContextCompat.startActivity(context, intent, null)
}}
```

LOGINACTIVITY2.KT

```
package com.example.podcastapplication

import android.os.Bundle

import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview

import com.example.podcastapplication.ui.theme.PodcastApplicationTheme

class loginActivity2 : 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
) {
    Greeting3("Android")
}
}
}
}
}
```

```
@Composable
fun Greeting3(name: String) {
    Text(text = "Hello $name!")
}
```

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview3() {
    PodcastApplicationTheme {
        Greeting3("Android")
    }
}
```

REGISTRATIONACTIVITY.KT

```
package com.example.podcastapplication

import android.os.Bundle

import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.example.podcastapplication.ui.theme.PodcastApplicationTheme
```

```
class registrationActivity : 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
                ) {
                    Greeting4("Android")
                }
            }
        }
    }
}
```

```

}

@Composable
fun Greeting4(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview4() {
    PodcastApplicationTheme {
        Greeting4("Android")
    }
}

```

ANDORIDMAIFEST.XML

```

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.PodcastApplication"

```

```
tools:targetApi="31">
<activity
    android:name=".registrationActivity"
    android:exported="false"
    android:label="@string/title_activity_registration"
    android:theme="@style/Theme.PodcastApplication" />
<activity
    android:name=".loginActivity2"
    android:exported="false"
    android:label="@string/title_activity_login2"
    android:theme="@style/Theme.PodcastApplication" />
<activity
    android:name=".Loginactivity1"
    android:exported="false"
    android:label="@string/title_activity_loginactivity1"
    android:theme="@style/Theme.PodcastApplication" />
<activity
    android:name=".loginactivity"
    android:exported="false"
    android:label="@string/title_activity_loginactivity"
    android:theme="@style/Theme.PodcastApplication" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

```
</intent-filter>

<activity
    android:name=".MainActivity"
    android:exported="true"
    android:label="@string/app_name"
    android:theme="@style/Theme.PodcastApplication">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
</manifest>
```

IC_LAUNCHER_BACKGROUND.xml

```
<?xml version="1.0" encoding="utf-8"?>

<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path
        android:fillColor="#3DDC84"
        android:pathData="M0,0h108v108h-108z" />
    <path
```

```
android:fillColor="#00000000"
android:pathData="M9,0L9,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M19,0L19,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M29,0L29,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M39,0L39,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M49,0L49,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```

```
android:fillColor="#00000000"
android:pathData="M59,0L59,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M69,0L69,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M79,0L79,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M89,0L89,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M99,0L99,108"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```



```
android:fillColor="#00000000"
android:pathData="M0,9L108,9"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,19L108,19"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,29L108,29"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,39L108,39"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,49L108,49"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```

```
android:fillColor="#00000000"
android:pathData="M0,59L108,59"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,69L108,69"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,79L108,79"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,89L108,89"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M0,99L108,99"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```

```
android:fillColor="#00000000"
android:pathData="M19,29L89,29"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M19,39L89,39"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M19,49L89,49"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M19,59L89,59"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M19,69L89,69"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```

```
android:fillColor="#00000000"
android:pathData="M19,79L89,79"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M29,19L29,89"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M39,19L39,89"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M49,19L49,89"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
android:fillColor="#00000000"
android:pathData="M59,19L59,89"
android:strokeWidth="0.8"
android:strokeColor="#33FFFFFF" />
<path
```

```
android:fillColor="#00000000"  
android:pathData="M69,19L69,89"  
android:strokeWidth="0.8"  
android:strokeColor="#33FFFFFF" />  
<path  
android:fillColor="#00000000"  
android:pathData="M79,19L79,89"  
android:strokeWidth="0.8"  
android:strokeColor="#33FFFFFF" />  
</vector>
```