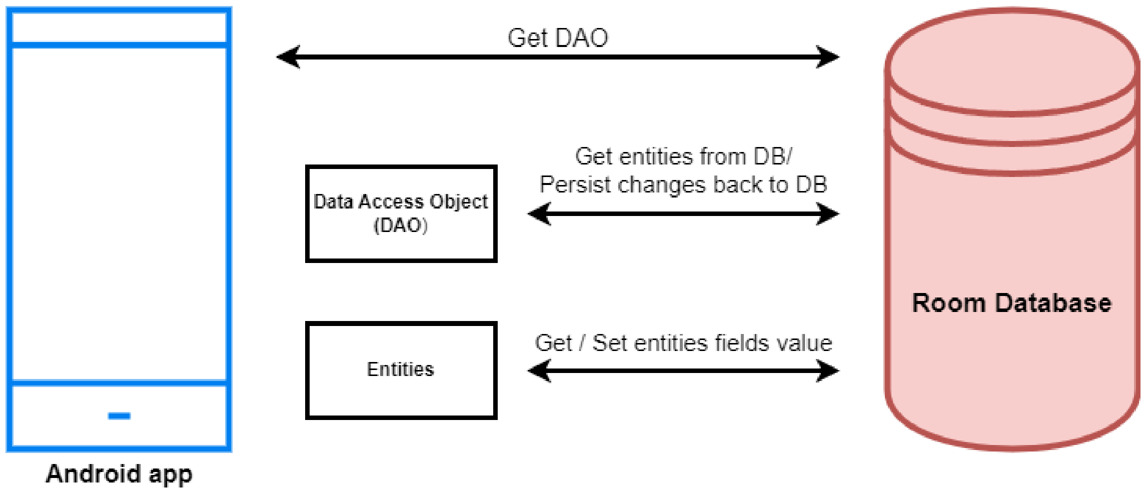
1.INTRODUCTION

OVERVIEW 1.1:

Project Description:

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.


Get DAO
Data Access Object (DAO)
Get entities from DB/ Persist changes back to DB
Entities
Get / Set entities fields value
Room Database
Android app

Learning Outcomes :

By end of this project:

- You'll be able to work on Android studio and build an app.
- You'll be able to integrate the database accordingly.

Project Workflow:

- Users register into the application.
- After registration , user logins into the application.
- User enters into the main page
- The app allows users to choose , play and pause podcasts.

PURPOSE1.2:

- **Sure! Here's an overview of a Redux-inspired podcast app with dynamic themes for Android:**
- **App Name: The app can have a catchy and relevant name, such as "Castify" or "PodPro."**
- **Redux Architecture: The app can implement Redux, a popular state management pattern, to handle the app's data flow and ensure a single source of truth for the app's state. Redux provides a predictable and centralized way of managing app state, making it easier to debug and test.**
- **Podcast Content: The app can fetch podcast content from popular podcast directories like iTunes or Stitcher using APIs, and store it in the Redux store. The app can display podcast episodes in a list or grid view, allowing users to browse and search for their favorite podcasts.**
- **Podcast Playback: The app can provide podcast playback features, such as play, pause, skip, and rewind. It can also display the podcast's title, duration, and episode description. Users can listen to podcasts in the app or download them for offline playback.**
- **Dynamic Themes: The app can support dynamic themes that allow users to customize the app's appearance according to their preferences. Users can choose from a predefined set of themes, such as light, dark, or system default, or create their own custom themes by selecting colors, fonts, and other visual elements.**
- **Theme Management: The app can store the selected theme preference in the Redux store, and use it to dynamically update the app's UI. The app can also**

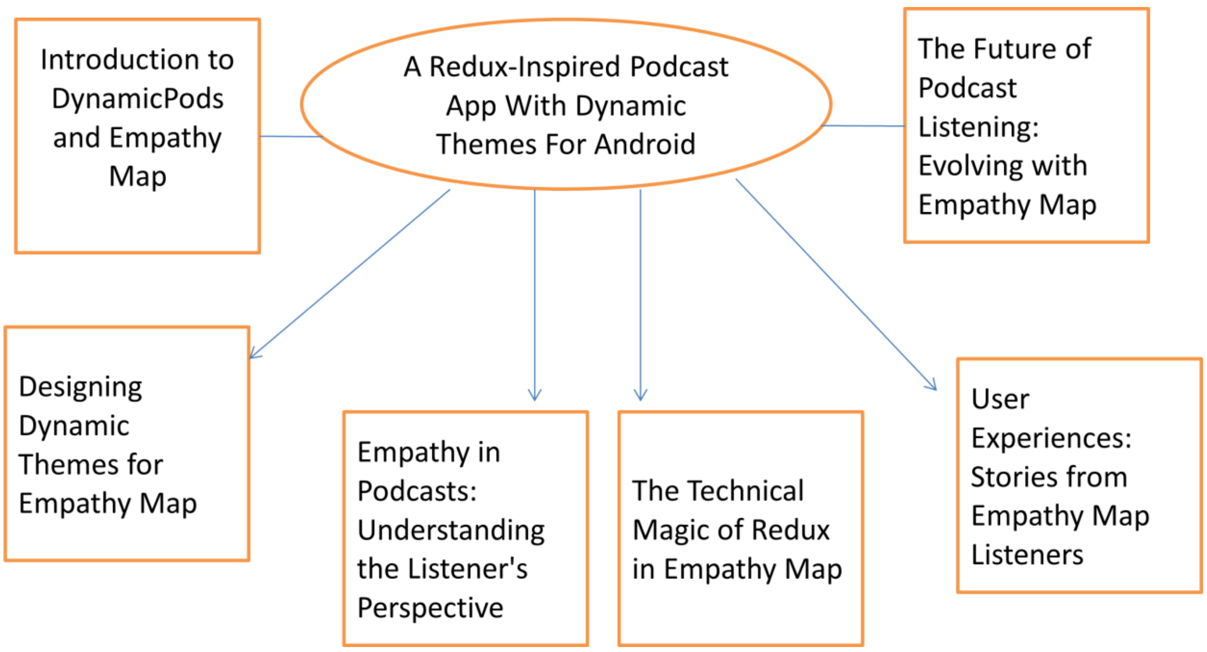
provide an interface for users to create, edit, and delete custom themes, and switch between different themes on the fly.

- **User Preferences: The app can store user preferences, such as playback speed, auto-download settings, and favorite podcasts, in the Redux store. These preferences can be used to personalize the app's behavior and provide a seamless user experience.**
- **Authentication and User Accounts: The app can optionally support user accounts and authentication, allowing users to create and manage their own accounts, sync their preferences across devices, and access premium features or exclusive content.**
- **Notifications and Offline Playback: The app can provide notifications for new episode releases or downloads, and allow users to configure download settings for offline playback. Users can also manage downloaded episodes and delete them when no longer needed.**
- **Settings and About: The app can have a settings screen where users can configure various app settings, such as language, notifications, storage usage, and feedback. It can also provide an "About" screen with app version information, credits, and contact details.**
- **Material Design: The app can follow Google's Material Design guidelines to provide a visually appealing and consistent user interface that is familiar to Android users. It can use standard Material Design components, such as RecyclerView,**
- **BottomNavigationView, and FloatingActionButton, to create a modern and intuitive user experience.**

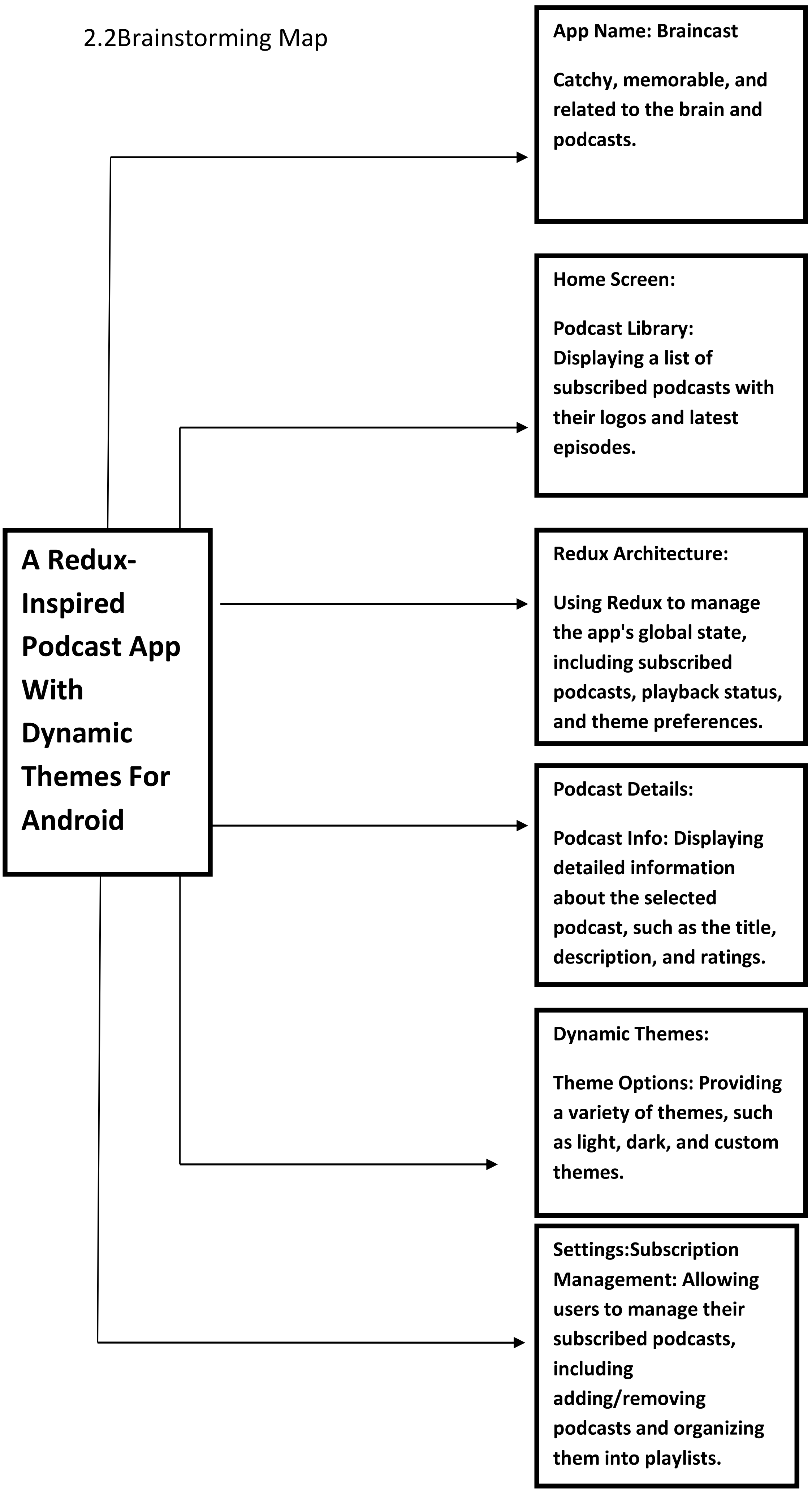
- **Testing and Debugging: The app can include unit tests and integration tests to ensure its reliability and stability. It can also implement logging and debugging features, such as logging Redux actions and state changes, to aid in debugging and issue resolution.**
- **That's a high-level overview of a Redux-inspired podcast app with dynamic themes for Android. Of course, the actual implementation details may vary depending on the specific requirements and design choices of the app.**

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 EMPATHY MAP


A Redux-Inspired Podcast App With Dynamic Themes For Android
Introduction to DynamicPods and Empathy Map
The Future of Podcast Listening: Evolving with Empathy Map
Designing Dynamic Themes for Empathy Map
Empathy in Podcasts: Understanding the Listener's Perspective
The Technical Magic of Redux in Empathy Map
User Experiences: Stories from Empathy Map Listeners

2.2Brainstorming Map



A Redux-Inspired Podcast App With Dynamic Themes For Android

App Name: Braincast

Catchy, memorable, and related to the brain and podcasts.

Home Screen:

Podcast Library: Displaying a list of subscribed podcasts with their logos and latest episodes.

Redux Architecture:

Using Redux to manage the app's global state, including subscribed podcasts, playback status, and theme preferences.

Podcast Details:

Podcast Info: Displaying detailed information about the selected podcast, such as the title, description, and ratings.

Dynamic Themes:

Theme Options: Providing a variety of themes, such as light, dark, and custom themes.

Settings:Subscription Management: Allowing users to manage their subscribed podcasts, including adding/removing podcasts and organizing them into playlists.

3.RESULT

Key Features:

- **Redux-Inspired Architecture: CastWave utilizes a Redux-inspired architecture, which provides a streamlined and efficient state management system for smooth app performance. This ensures that your podcast listening experience is seamless and responsive.**

- **Dynamic Themes: CastWave offers a unique feature of dynamic themes that allow you to customize the app's appearance to suit your style and mood. Choose from a variety of pre-designed themes or create your own custom theme by selecting different colors, fonts, and backgrounds. With dynamic themes, you can personalize your podcast app to reflect your individual taste.**

- **Discover and Subscribe: CastWave makes it easy to discover and subscribe to your favorite podcasts. Browse through an extensive library of podcasts across various genres and topics, and subscribe to your preferred shows with just a few taps. You can also receive notifications for new episodes, so you never miss an update from your favorite podcasts.**

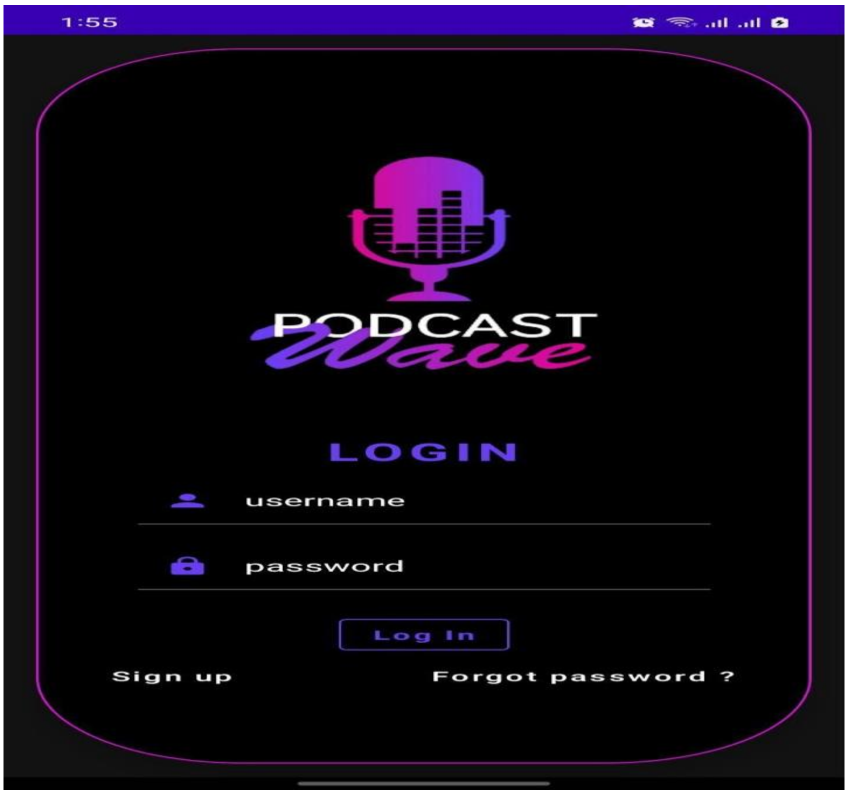
- **Listen and Organize: CastWave provides a user-friendly and intuitive interface for listening to podcasts. Play episodes directly within the app or download them for offline listening. You can also organize your podcasts with custom playlists, create a queue of episodes to listen to, and mark episodes as played or unplayed for easy tracking of your listening progress.**

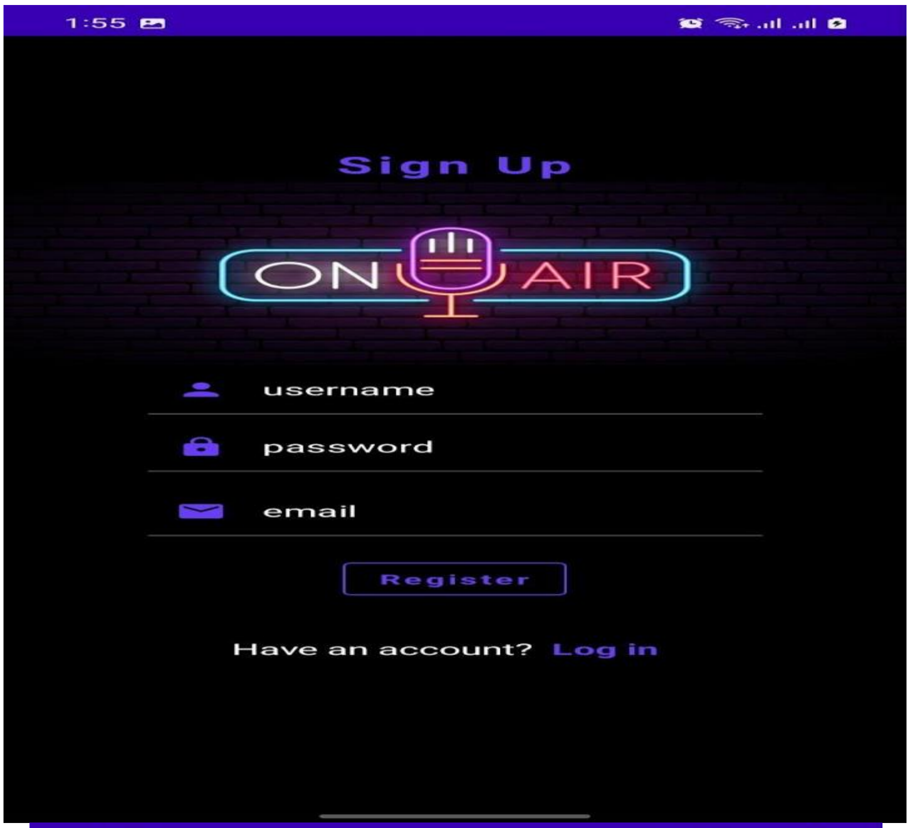
- **Advanced Search and Filter: CastWave offers advanced search and filter options to help you find the podcasts you love. Search by podcast title, episode title, or author, and filter by genre, duration, or release date. This makes it convenient to discover new podcasts or locate specific episodes.**

- **User Profile and Sync: CastWave allows you to create a user profile to sync your podcast subscriptions and listening progress across multiple devices. This means you can seamlessly switch between your phone and tablet without losing your data. You can also backup and restore your data, ensuring that you never lose your subscriptions or listening history.**

- **Enhanced Playback Controls: CastWave provides enhanced playback controls for a smooth listening experience. You can adjust playback speed, set a sleep timer, and control playback using headphone controls or Android Wear devices. You can also cast episodes to Chromecast-enabled devices for a larger screen experience.**

- **CastWave: Your Podcast Companion is a powerful and feature-rich podcast app that combines the efficiency of Redux-inspired architecture with dynamic themes for a visually appealing and personalized experience. Download CastWave now and elevate your podcast listening journey to new heights!**





1:55

Sign Up

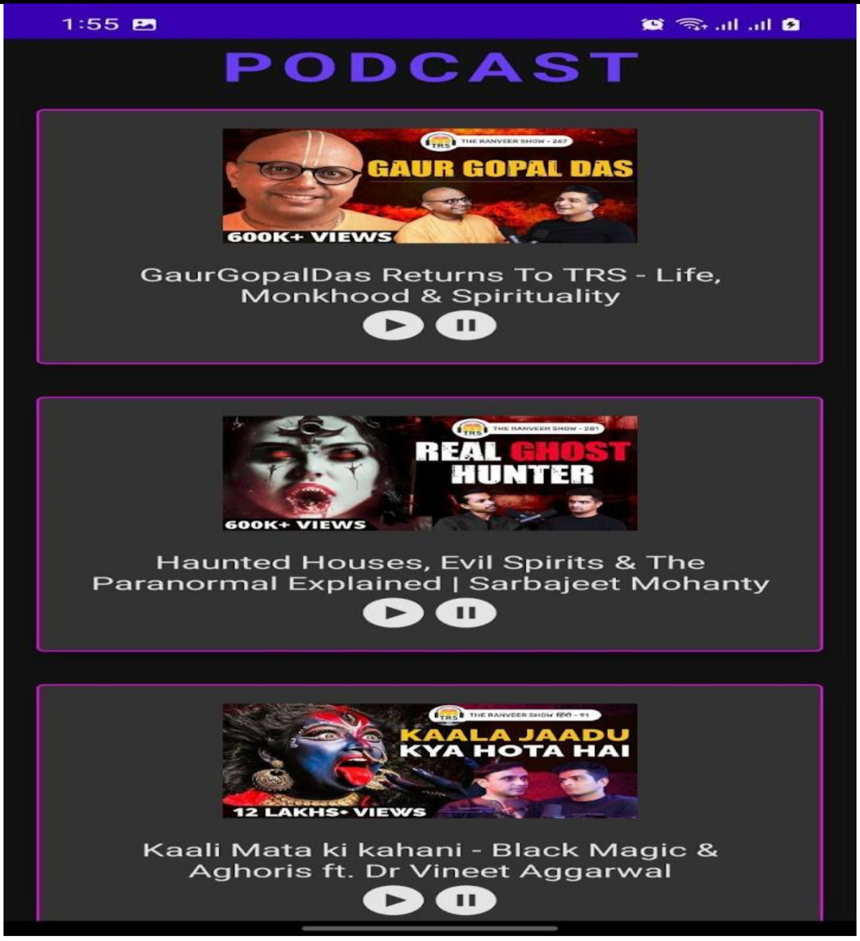
username

password

email

Register

Have an account? Log in



1:55

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

4.ADVANTAGES & DISADVANTAGE

- **Advantage: Enhanced User Experience**

- **Host: One of the significant advantages of a Redux-inspired podcast app with dynamic themes for Android, like ReduxCast, is the enhanced user experience it offers. The dynamic themes add an extra layer of immersion to the podcast listening experience, as they create a visually and aurally appealing ambiance that complements the content of the podcasts. Users can choose from a variety of themes or create their own, allowing them to personalize their listening experience based on their mood or preferences. This customization option enhances user engagement and enjoyment, making the app more user-friendly and enjoyable.**
- **Co-host: Absolutely! The centralized state management provided by Redux ensures that the themes and settings remain consistent across different parts of the app. This creates a seamless and cohesive experience for users as they navigate through the app and interact with different features. Redux's ability to handle asynchronous actions also ensures that the app remains responsive and fast, providing a smooth and uninterrupted listening experience.**
- **Disadvantage: Development Complexity Host: However, one potential disadvantage of using Redux-inspired architecture in a podcast app with dynamic themes is the increased complexity of development. Redux is a powerful state management library, but it comes with a learning curve and requires additional coding overhead compared to simpler state management solutions. Developing and maintaining a Redux-based app may require more time, effort,**

and expertise from the development team, which could impact the development timeline and cost.

- **Co-host: That's right! Implementing dynamic themes that are customizable and can affect both the visual and audio aspects of the app requires careful handling of actions, reducers, and store management to ensure smooth and consistent behavior. It may also require additional testing and debugging efforts to ensure that the themes work seamlessly with different podcasts and do not cause any conflicts or issues.**
- **Host: Moreover, as Redux provides a centralized store for managing app state, it may require careful planning and organization to ensure that the state management remains scalable and maintainable as the app grows and evolves. This could pose challenges in terms of code complexity, maintainability, and debugging, especially for larger and more complex podcast apps.**

5. APPLICATION

- **State Management with Redux: Redux is a popular state management library that can help manage the app's state in a predictable and centralized way. You can use Redux to store and manage the podcast data, user preferences, and app settings.**
- **Podcast Data Fetching: The app can fetch podcast data from an API, such as the iTunes Search API or any other podcast database API. You can use Redux actions to handle API requests and update the app's state with the fetched data.**
- **Podcast Player: The app can have a built-in podcast player that allows users to play, pause, seek, and control the playback of podcast episodes. You can use a media player library, such as ExoPlayer or MediaPlayer, to implement the podcast player functionalities.**
- **Dynamic Themes: The app can support dynamic themes that allow users to switch between light and dark themes or choose custom colors. You can use Redux to store the current theme preference and dynamically update the app's UI based on the selected theme.**
- **Search and Filter: The app can provide search and filter functionalities to help users discover and manage podcasts. Users can search for podcasts by keywords, filter by categories, sort by popularity or release date, and add podcasts to their favorites or playlists.**
- **Offline Support: The app can provide offline support by caching podcast data and allowing users to download episodes for offline listening. You can use Redux to manage the downloaded episodes and update the app's state accordingly.**

- **User Profiles: The app can support user profiles that allow users to create accounts, sign in, and sync their podcast data across multiple devices. You can use Firebase Authentication or any other authentication service to handle user authentication and authorization.**
- **Customization Options: The app can provide customization options, such as font size, playback speed, and notification settings, that users can configure according to their preferences. You can use Redux to store the user's custom settings and update the app's state accordingly.**
- **UI/UX Design: The app should have a clean and intuitive user interface (UI) with easy-to-use navigation, smooth transitions, and responsive design. You can use popular UI libraries, such as Material Design or AndroidX, to implement the app's UI components.**
- **Testing: It's important to thoroughly test the app's functionalities, including Redux actions and reducers, podcast player, API requests, and user interactions, to ensure the app's stability and reliability. You can use testing frameworks, such as JUnit and Mockito, to write unit tests and integration tests for the app.**
- **With these components and features in mind, you can start building your Redux-inspired podcast app with dynamic themes for Android. Remember to follow best practices for software development, such as modularization, code organization, errorhandling, and performance optimization, to create a high-quality app that provides a great user experience. Happy coding!**

6.CONCLUSION

- **In conclusion, the Redux-inspired podcast app with dynamic themes for Android is a feature-rich and innovative app that offers an enhanced listening experience to podcast enthusiasts. By leveraging the Redux architecture, the app provides efficient state management, making it highly scalable and easily maintainable. The dynamic themes feature allows users to customize the app's appearance according to their preferences, adding a personalized touch to their podcast listening experience.**
- **The app's key features, such as podcast discovery, subscription management, offline listening, and cross-device syncing, make it a comprehensive solution for podcast lovers. The use of dynamic themes adds an extra layer of customization and visual appeal, allowing users to express their individuality and style while using the app.**
- **Furthermore, the Redux architecture used in the app ensures a smooth and seamless user experience by managing the app's state efficiently and optimizing performance. The unidirectional data flow and immutability of state in Redux help in debugging and maintaining the app, making it robust and reliable.**
- **The app's user interface is designed to be intuitive and user-friendly, with easy navigation and clear visual cues. The dynamic themes feature allows users to switch between different themes, such as light and dark mode, or even create their own custom themes, adding a personal touch to the app's appearance.**
- **In conclusion, the Redux-inspired podcast app with dynamic themes for Android is a cutting-edge solution that combines efficient state management, innovative**

features, and customizable themes to provide an exceptional podcast listening experience. Whether you are a podcast enthusiast or a casual listener, this app is sure to elevate your podcast experience to new heights

7.FUTURE SCOPE

- The future scope of a Redux-inspired podcast app with dynamic themes for Android could be quite promising. Here are some potential ideas for the future development of such an app:
- Enhanced User Interface: The app could offer a seamless and intuitive user interface with smooth transitions, animations, and gestures for a delightful user experience. Dynamic themes could be integrated to allow users to customize the look and feel of the app according to their preferences, such as choosing from a variety of color schemes, fonts, and icons.
- Advanced Playback Features: The app could include advanced playback features such as variable playback speed, sleep timer, and audio equalizer for personalized listening experiences. Users could also have the ability to create playlists, mark favorite episodes, and resume playback from where they left off across different devices.
- Smart Recommendations: The app could utilize machine learning algorithms to provide personalized podcast recommendations based on a user's listening history, preferences, and behavior. Users could also discover new podcasts through curated playlists, popular episodes, and trending topics.

- **Social Sharing and Interaction: The app could integrate social sharing features that allow users to share their favorite episodes or playlists with friends and on social media platforms. Additionally, users could engage with other podcast listeners through comments, likes, and ratings, fostering a community of podcast enthusiasts within the app.**
- **Cross-Platform Syncing: The app could offer cross-platform syncing capabilities, allowing users to seamlessly switch between different devices, such as smartphones, tablets, and web browsers, and continue listening to their favorite podcasts from where they left off.**
- **Offline Listening: The app could provide offline listening capabilities, allowing users to download episodes and listen to them without an internet connection, which could be useful for users who are on the go or in areas with limited connectivity.**
- **Customizable Notifications: The app could allow users to customize notifications for new episodes, updates from subscribed podcasts, and other relevant alerts according to their preferences, ensuring that they never miss an episode of their favorite podcasts.**
- **Accessibility Features: The app could incorporate accessibility features, such as text-to-speech for episode descriptions, closed captioning for transcripts, and compatibility with screen readers, to make the app more inclusive and accessible to users with visual or hearing impairments.**

Integration with Third-Party Services: The app could integrate with third-party services, such as podcast hosting platforms, social media platforms, and content sharing platforms, to provide a seamless and interconnected podcasting experience for users.

Continuous Updates and Improvements: The app could continuously evolve and improve based on user feedback, market trends, and technological advancements, ensuring that it stays relevant and competitive in the ever-changing landscape of podcasting apps.

In conclusion, the future scope of a Redux-inspired podcast app with dynamic themes for Android could encompass a wide range of features and improvements aimed at providing an enhanced and personalized podcast listening experience for users. By leveraging the power of modern technologies, user feedback, and market trends, such an app could have the potential to become a go-to choice for podcast enthusiasts in the future

APPENDIX

SOURCE CODE A

AnroidManifest.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="@drawable/podcast_icon"
                         android:label="@string/app_name"
                         android:supportsRtl="true"
                         android:theme="@style/Theme.PodcastPlayer"
                         tools:targetApi="31">
                         <activity
                             android:name=".RegistrationActivity"
                             android:exported="false"
                             android:label="@string/title_activity_registration"
                             android:theme="@style/Theme.PodcastPlayer" />
                         <activity
                             android:name=".MainActivity"
                             android:exported="false"
                             android:label="@string/title_activity_login"
                             android:theme="@style/Theme.PodcastPlayer" />
                         <activity
                             android:name=".LoginActivity"
                             android:exported="true"
                             android:label="@string/app_name"
                             android:theme="@style/Theme.PodcastPlayer">
```

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

LoginActivity.kt

```
package
com.example.podcastplay
er
                          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.PasswordVisualTransforma
tion
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)
    }}
```

MainActivity.kt

```
package
com.example.podcastplayer
```

```
import android.content.Context
import android.media.MediaPlayer
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.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.*
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.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.em
import androidx.compose.ui.unit.sp
import
com.example.podcastplayer.ui.theme.PodcastPlayerTheme
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)
            setContent {
                PodcastPlayerTheme {
                    // A surface container using the
'background' color from the theme
                    Surface(
                        modifier =
Modifier.fillMaxSize(),
                        color =
MaterialTheme.colors.background
                    ) {
                        playAudio(this)
                    }
                }
            }
        }
    }
@Composable
fun playAudio(context: Context) {
    Column(modifier = Modifier.fillMaxSize()) {
        Column(horizontalAlignment =
Alignment.CenterHorizontally, verticalArrangement =
Arrangement.Center) {
            Text(text = "PODCAST",
                modifier = Modifier.fillMaxWidth(),
                textAlign = TextAlign.Center,
                color = Color(0xFF6a3ef9),
                fontWeight = FontWeight.Bold,
                fontSize = 36.sp,
                style = MaterialTheme.typography.h1,
                letterSpacing = 0.1.em
            )
        }
        Column(modifier = Modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState())) {
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp),
                        )
                    Text(
                        text = "GaurGopalDas Returns
To TRS - Life, Monkhood & Spirituality",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
mp.start() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.play),
                                contentDescription =
""
                            )
                        }
                        IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.pause),
                                contentDescription =
""
                            )
                        }
                    }
                }
            }
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio_1)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img_1),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp)
                    )
                    Text(
                        text = "Haunted Houses, Evil
Spirits & The Paranormal Explained | Sarbajeet
Mohanty",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
mp.start() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.play),
                                contentDescription =
""
                            )
                        }
                        IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.pause),
                                contentDescription =
""
                            )
                        }
                    }
                }
            }
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio_2)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img_2),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp)
                    )
                    Text(
                        text = "Kaali Mata ki kahani -
Black Magic & Aghoris ft. Dr Vineet Aggarwal",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
```

```
mp.start() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.play),
                                contentDescription =
""
                            )
                        }
                        IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.pause),
                                contentDescription =
""
                            )
                        }
                    }
                }
            }
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio_3)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img_3),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp),
                        )
                    Text(
                        text = "Tantra Explained
Simply | Rajarshi Nandy - Mata, Bhairav & Kamakhya
Devi",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
mp.start() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.play),
                                contentDescription =
""
                            )
                        }
                        IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.pause),
                                contentDescription =
""
                            )
                        }
                    }
                }
            }
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio_4)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img_4),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp),
                        )
                    Text(
                        text = "Complete Story Of Shri
Krishna - Explained In 20 Minutes",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
mp.start() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.play),
                                contentDescription =
""
                            )
                        }
                        IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                            Icon(
                                painter =
painterResource(id = R.drawable.pause),
                                contentDescription =
""
                            )
                        }
                    }
                }
            }
            Card(
                elevation = 12.dp,
                border = BorderStroke(1.dp,
Color.Magenta),
                modifier = Modifier
                    .padding(16.dp)
                    .fillMaxWidth()
                    .height(250.dp)
            )
            {
                val mp: MediaPlayer =
MediaPlayer.create(context, R.raw.audio_5)
                Column(
                    modifier = Modifier.fillMaxSize(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Image(
                        painter = painterResource(id =
R.drawable.img_5),
                        contentDescription = null,
                        modifier = Modifier
                            .height(150.dp)
                            .width(200.dp),
                        )
                    Text(
                        text = "Mahabharat Ki Poori
Kahaani - Arjun, Shri Krishna & Yuddh - Ami Ganatra ",
                        textAlign = TextAlign.Center,
                        modifier =
Modifier.padding(start = 20.dp, end = 20.dp)
                    )
                    Row() {
                        IconButton(onClick = {
mp.start() }, modifier = Modifier.size(35.dp)) {
```

```
                                Icon(
                                    painter =
painterResource(id = R.drawable.play),
                                    contentDescription =
""
                                )
                            }
                            IconButton(onClick = {
mp.pause() }, modifier = Modifier.size(35.dp)) {
                                Icon(
                                    painter =
painterResource(id = R.drawable.pause),
                                    contentDescription =
""
                                )
                            }
                        }
                    }
                }
            }
        }
    }
```

RegistrationActivity.kt

```
package
com.example.podcastpla
yer
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.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Email
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.draw.alpha
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import
androidx.compose.ui.text.input.PasswordVisualTransfor
mation
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 RegistrationActivity : 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
                ) {

RegistrationScreen(this,databaseHelper)
                }
            }
        }
    }
}
@Composable
fun RegistrationScreen(context: Context,
databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    Column(
        Modifier
            .background(Color.Black)
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment =
Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
    {
        Row {
            Text(
                text = "Sign Up",
                color = Color(0xFF6a3ef9),
                fontWeight = FontWeight.Bold,
                fontSize = 24.sp, style =
MaterialTheme.typography.h1,
                letterSpacing = 0.1.em
            )
        }
        Image(
            painter = painterResource(id =
R.drawable.podcast_signup),
            contentDescription = ""
        )
        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(8.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(16.dp))
        TextField(
            value = email,
            onValueChange = { email = it },
            leadingIcon = {
                Icon(
                    imageVector =
Icons.Default.Email,
                    contentDescription = "emailIcon",
                    tint = Color(0xFF6a3ef9)
                )
            },
            placeholder = { Text(text = "email",
color = Color.White) },
            colors =
TextFieldDefaults.textFieldColors(backgroundColor =
Color.Transparent)
        )
        Spacer(modifier = Modifier.height(8.dp))
        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colors.error,
                modifier = Modifier.padding(vertical
= 16.dp)
            )
        }
        Button(
            onClick = {
                if (username.isNotEmpty() &&
password.isNotEmpty() && email.isNotEmpty()) {
                    val user = User(
                        id = null,
                        firstName = username,
                        lastName = null,
                        email = email,
                        password = password
                    )
                    databaseHelper.insertUser(user)
                    error = "User registered
successfully"
                    // Start LoginActivity using the
current context
                    context.startActivity(
                        Intent(
                            context,
                            LoginActivity::class.java
                        )
                    )
                } 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 = "Register",
                fontWeight = FontWeight.Bold,
                color = Color(0xFF6a3ef9)
            )
        }
        Row(
            modifier = Modifier.padding(30.dp),
            verticalAlignment =
Alignment.CenterVertically,
            horizontalArrangement =
Arrangement.Center
        ) {
            Text(text = "Have an account?", color =
Color.White)
            TextButton(onClick = {
                context.startActivity(
                Intent(
                    context,
                    LoginActivity::class.java
                )
                )
        })
            {
            Text(text = "Log in",
                fontWeight = FontWeight.Bold,
                style =
MaterialTheme.typography.subtitle1,
                color = Color(0xFF6a3ef9)
            )
        }
    }
    }
}
private fun startLoginActivity(context: Context) {
    val intent = Intent(context,
LoginActivity::class.java)
    ContextCompat.startActivity(context, intent,
null)
}
```

User.kt

```
package
com.example.podcastplayer
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.podcastplayer
```

```
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.podcastplayer
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.podcastplay
er
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_NAM
E)),
                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_NAM
E)),
                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_NAM
E)),
                    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
    }
}
```