Android Application Development

Podcast plus: A redux-inspired podcast app with dynamic themes for android

SUMAN K	028A6ABAA6A5C08B52C42F68D9E456E2
RAHUL R	827C598992A9B33BAD74300391183F11
SABARI GANESAN K	35E8CDF3A060E868C394BEB65F03346B
MADESH K	B25630204B644898BAB41844052EB5A9

SEMESTER – V
B.TECH ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR-2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE COIMBATORE-641046

NOVEMBER 2024

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

INTRODUCTION:

The Podcast Plus app is an innovative podcast player designed for Android devices, aimed at providing an intuitive, feature-rich experience for podcast listeners. This app leverages modern Android development tools and architectural principles to offer users a seamless and interactive platform for discovering, playing, and managing their favorite podcasts. Built using Jetpack Compose, a declarative UI toolkit for Android, the app ensures that the user interface is not only efficient but also easy to maintain and extend.

One of the key features of the Podcast Plus app is its dynamic theming capability, which allows users to toggle between light and dark modes according to their preferences. This enhances user experience, as dynamic themes can contribute to better readability and accessibility, depending on the user's environment and time of day.

The app is developed using a Redux-style architecture, which is an implementation of a unidirectional data flow. This approach helps manage the application's state in a predictable and scalable way, which is especially useful when dealing with complex UI interactions such as podcast playback, user preferences, and theme settings. The Redux architecture is composed of actions, reducers, and a store, making it easier to track and update state across different parts of the app.

Furthermore, the app includes offline data storage capabilities, implemented using Room, Android's official database library. This allows users to enjoy their podcasts even when they don't have an active internet connection. The app can cache podcasts, user information, and preferences, ensuring that users can continue their podcast experience without interruption.

Throughout this project, developers will gain hands-on experience in building a full-featured Android app that combines state management, dynamic theming, local data persistence, and modern UI development practices. By following the architecture and workflow outlined in this project, developers will not only build a functional podcast player but will also acquire the skills to develop other Android applications with similar requirements.

The Podcast Plus app aims to offer a simple yet highly functional podcasting solution while also demonstrating advanced Android app development practices, enabling developers to create apps that are scalable, maintainable, and user-friendly.

OBJECTIVES:

1. User Registration and Login

The app allows users to register and log in securely. Upon registration, users create an account with a username and password. Once logged in, they can access personalized features like saved podcasts, playback history, and preferences. This ensures a customized experience tailored to each user.

2. Explore and Play Podcasts

Users can browse a wide variety of podcasts, organized by categories such as technology, comedy, and education. The app provides a simple yet powerful interface for selecting podcasts, viewing detailed information, and playing episodes. Basic playback controls such as play, pause, and skip are included to give users full control over their listening experience.

3. Jetpack Compose for UI Development

The user interface of the app is built using Jetpack Compose, Android's modern toolkit for building UIs. Jetpack Compose enables a declarative approach to UI design, making the app's layout more efficient and easier to maintain. UI elements will automatically update in response to changes in the app's state, providing a smooth and dynamic user experience.

4. Redux-Style Architecture for State Management

The app uses a Redux-style architecture, following a unidirectional data flow. This design helps manage the app's state in a predictable and scalable way, ensuring that actions such as podcast playback and theme switching are consistently reflected across the app. This architecture includes actions, reducers, and a central store, which simplifies state updates and debugging.

6. Local Database Integration

The app integrates a local database using Room, Android's official database library, to store podcasts and user preferences. This allows users to continue listening to podcasts even when offline. The database will store podcast metadata, playback progress, and user preferences such as theme selection, ensuring that users can seamlessly pick up where they left off, even after restarting the app.

SYSTEM REQUIREMENTS TO BUILD THE APP

1. Hardware Requirements

- Computer/Workstation:
  - Processor: Minimum Intel i5 or equivalent (Quad-core processor recommended).
  - RAM: 8 GB of RAM (16 GB recommended for better performance with Android Studio and other tools).
  - Storage: At least 100 GB of free disk space (more may be required as you add resources like images, videos, and database files).
  - Display: A monitor with at least a Full HD resolution (1920x1080) for optimal viewing of code and UI design in Android Studio.

2. Software Requirements

- Operating System:
  - Windows: Windows 10 (64-bit) or higher
  - macOS: macOS Mojave (10.14) or higher
  - Linux: Ubuntu 20.04 LTS or equivalent
- Android Development Environment:
  - Android Studio: The official Integrated Development Environment (IDE) for Android development.
    - Version: Android Studio 2020.3.1 or higher (recommended for the latest features and compatibility with Jetpack Compose).
    - SDK: Android SDK (Software Development Kit) installed with required APIs for Android 8.0 (Oreo) or higher.
    - JDK: Java Development Kit (JDK) version 11 or higher.
- Programming Languages:
  - Kotlin: The primary programming language used for the development of Android applications.
  - XML: For defining layouts, though Jetpack Compose will minimize XML usage.

- Libraries and Frameworks:
    - Jetpack Compose: The modern toolkit for building native UIs in Android.
    - Room Database: For local data storage, including podcasts and user preferences.
    - Retrofit or Volley: For handling network requests (e.g., fetching podcast data from a remote server).
    - Firebase Authentication (optional): If using Firebase for user authentication and session management.
- Version Control:
    - Git: For version control and collaborating on code using Git repositories.
    - GitHub or GitLab: Optional, for remote code hosting and project management.

3. Network and Internet Requirements

- Internet Connection: A stable internet connection is required for:
    - Accessing external libraries and dependencies (via Gradle).
    - Downloading Android SDK updates or new versions of Android Studio.
- API/Server (Optional):
    - If you are using a custom podcast API or any other external API (e.g., iTunes or Spotify API), a RESTful API or GraphQL API must be set up for podcast data retrieval.

4. Development Tools

- Android Emulator: Part of Android Studio, used for simulating Android devices for testing the app without needing physical devices. Ensure you have an emulator image for Android 8.0 (Oreo) or higher.
- Device for Testing: While the Android Emulator is sufficient, testing the app on a physical Android device (e.g., phone or tablet) is recommended for real-world performance testing.
- Gradle: A build automation tool used by Android Studio to handle project dependencies and manage builds.

SYSTEM REQUIREMENTS TO USE THE APP

1. Hardware Requirements

To run the Podcast Plus app on Android devices, the following hardware specifications are recommended:

- Device: Android smartphones or tablets
    - Minimum Screen Size: 4.5 inches (recommended 5.5 inches or larger for better user experience)
    - Processor: ARM-based processor, with at least a quad-core 1.2 GHz CPU for smooth performance
    - RAM: Minimum 2 GB RAM, 4 GB RAM or higher recommended for smoother performance and multitasking
    - Storage: At least 50 MB of free internal storage for the app installation; additional storage may be required if users wish to download podcasts for offline listening
    - Battery: Standard Android battery, though prolonged listening (especially with background playback) will require adequate battery capacity
    - Audio Output: Built-in speakers or connected Bluetooth headphones/speakers for podcast audio playback

2. Software Requirements

The Podcast Plus app is compatible with Android devices running the following software:

- Operating System:
    - Android 8.0 (Oreo) or higher (recommended Android 10 or higher for the best performance and compatibility with features like dark mode, Jetpack Compose, and more).
- App Dependencies:
    - Google Play Services: Required for services such as authentication (if using Firebase) and cloud storage (if supported).
    - Google Play Store: Users should be able to install the app from the Google Play Store (or sideload it, though the former is recommended).

- Network Connectivity:
    - Offline Mode: While an internet connection is required for streaming new content, podcasts that are downloaded to the device will be available for offline listening.
    - Internet Speed: A stable connection (minimum 3G or faster) is recommended for seamless podcast streaming. 4G or Wi-Fi is preferred for optimal streaming experience.

3. Permissions

- Storage Access: Required for downloading and storing podcast episodes locally for offline listening.
- Network Access: To stream podcasts and synchronize data with external APIs or the app’s cloud services.
- Audio Output: Access to the device’s audio system for podcast playback through speakers, headphones, or Bluetooth devices.
- Background Activity: Necessary to continue podcast playback when the app is in the background or the device screen is turned off.

4. Additional Requirements

- Headphones/External Speakers: For enhanced audio quality during podcast playback.
- Bluetooth Support: For users who prefer listening to podcasts via Bluetooth headphones or car audio systems.

5. App Installation and Updates

- Installation: The app can be installed via the Google Play Store. Users with a compatible Android device and operating system can download the app directly from the Play Store.
- Updates: The app will periodically receive updates to add new features, fix bugs, and improve performance. A stable internet connection is needed to download and install updates.

KEY FUNCTIONALITIES

User Registration and Login

The Podcast Plus app begins with an essential user registration process, ensuring a personalized experience for each user. Users are first prompted to register by providing their username and password, which are then securely stored for future authentication. During registration, the app may include optional fields, such as email or preferences, to tailor the experience further.

Upon successful registration, users can proceed to log into the app. The login process involves entering the previously created credentials (username and password) to authenticate their identity. User authentication is managed using the app's internal state, which communicates with the backend or a local authentication service (e.g., Firebase Authentication or a custom server). Once authenticated, users are granted access to the main page of the app.

The main goal of this flow is to ensure that only registered users can access and interact with their personalized podcast content, creating a secure and tailored experience. After logging in, users are taken directly to the main page, which serves as the hub for exploring and managing podcasts.

Main Page

- Podcast Categories: The podcasts are organized into categories such as Technology, Education, Comedy, etc., allowing users to browse content that suits their interests. This categorization makes it easy for users to discover new shows and episodes.
- Podcast Information: Each podcast is displayed with basic metadata, including its name, cover image, and a brief description. Users can quickly get an overview of the podcast before deciding to listen. Detailed episode information is available when a user taps on a podcast, allowing them to select specific episodes to play.
- Playback Controls: The main page features intuitive play/pause, skip, and resume buttons for managing podcast playback. These controls allow users to seamlessly switch between podcasts, pause playback, and resume from where they last left off.
- Podcast Search: A search bar may also be available on the main page to help users quickly find specific podcasts or episodes by title, category, or keyword.

Podcast Control Features

- Select a Podcast: From the list of available podcasts, users can browse or search for their favorite shows and tap on an episode to start playing. The app supports multiple episodes within each podcast, and users can select the episode they wish to listen to.
- Play/Pause: The play/pause button allows users to start or pause the podcast. This is a core feature, enabling users to take a break or continue listening at their convenience.
- Stop/Start Playback: The podcast player interface also includes a stop or restart button. This allows users to completely stop playback and return to the beginning of the episode. It is useful when users want to re-listen to an episode or switch between episodes.
- Seek Bar: A seek bar enables users to fast-forward or rewind the podcast to a specific point in the episode. This gives users full control over their listening experience, ensuring they can quickly navigate through the content.

Dynamic Themes

A key feature of the Podcast Plus app is its support for dynamic themes. Users can toggle between a light theme and a dark theme, depending on their preferences and environmental lighting conditions. This feature enhances user comfort, as many users prefer dark mode in low-light environments or lighter themes in well-lit settings.

- Theme Settings: The app provides an option in the settings menu for users to toggle between light and dark modes. This setting is easily accessible and can be changed at any time, giving users full control over their app experience.
- Persistent Theme Preference: Once a user selects their preferred theme, the choice is persisted within the app’s settings. This means that the next time the user opens the app, it will automatically load in the selected theme without requiring them to make the change again. This persistent experience ensures that users don’t have to reset their preferences after each session.
- Smooth Transition: The transition between themes is smooth and immediate, providing a visually seamless experience when switching between modes. The app uses Jetpack Compose to ensure that the UI components and elements dynamically adjust to the selected theme.

FUTURE ENHANCEMENTS

1. Cross-Platform Expansion

Expanding Podcast Plus to iOS would allow the app to reach a broader audience. By developing an iOS version, the app can engage users across both major mobile platforms, providing a consistent experience and taking advantage of features like Siri integration and Apple Podcasts compatibility.

2. Social Features

Integrating social sharing options will enable users to share episodes and playlists on platforms like Facebook and Twitter, increasing app visibility. Additionally, adding community features like comments, ratings, and discussion forums will foster interaction among users and podcast creators, enhancing user engagement.

3. Enhanced Analytics

Implementing advanced user analytics will provide insights into user behavior, helping developers and creators improve content offerings. Tracking metrics like listening patterns, episode popularity, and user preferences will allow for better podcast recommendations and personalized user experiences.

4. Monetization Strategies

To support future development, premium features could be introduced, such as ad-free listening and exclusive content. A subscription model or one-time payment options could provide users with enhanced features while generating revenue for the app.

5. Live Streaming

Adding live podcast streaming would allow users to engage with real-time content, ask questions during live shows, and interact directly with creators. This feature could introduce a new level of interactivity and foster community engagement.

6. Voice Command Integration

Integrating voice command functionality would allow users to control playback hands-free (e.g., “Play the latest episode of [Podcast Name]”). This feature would make the app more accessible and user-friendly, especially when used in conjunction with voice assistants like Google Assistant and Siri.

PROGRAM

LOGIN ACTIVITY.KT

```
package 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("") }
    val hue = 193f / 360f          // Convert hue to a fraction of 360
    val saturation = 1.0f          // Saturation as a decimal (100%)
    val lightness = 0.92f          // Lightness as a decimal (92%)

    // Create Color from HSL
    val hslColor = Color.hsl(hue, saturation, lightness)
    Card(
        elevation = 12.dp,
        backgroundColor = hslColor,
        border = BorderStroke(1.dp, Color.hsl(hue,saturation,lightness)),
        shape = RoundedCornerShape(100.dp),
        modifier = Modifier.padding(16.dp).fillMaxWidth()
    ) {
        Column(
            Modifier
                .background(hslColor)
                .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(0xFF67DA00),
    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(0xFF67DA00)
        )
    },
    placeholder = {
        Text(
            text = "username",
            color = Color.Black
```

```
                )
            },
            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(0xFF67DA00)
                )
            },
            placeholder = { Text(text = "password", color = Color.Black) },
            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(0xFF04B445)),
            colors = ButtonDefaults.buttonColors(backgroundColor =
Color.Black),
            modifier = Modifier.padding(top = 16.dp)
        ) {
            Text(text = "Log In", fontWeight = FontWeight.Bold, color =
Color(0xFF04B445))
        }

        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.Black
                )
            }
        }
    }
}

    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(0xFFF7B5CA),
                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.Black),
                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 = "Everyday is a chance to change yourself",
                        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 = "LISTEN When you wake up 10 minutes to start your day
            right",
                                    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 = "Push Yourself because no one else is going to do it For
You",
                    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 = "Take it easy | RJ balaji",
                    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 = "True Success requires sacrifice",
                    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 = ""
                        )
                    }
                }
            }

        }
    }
}
}
```

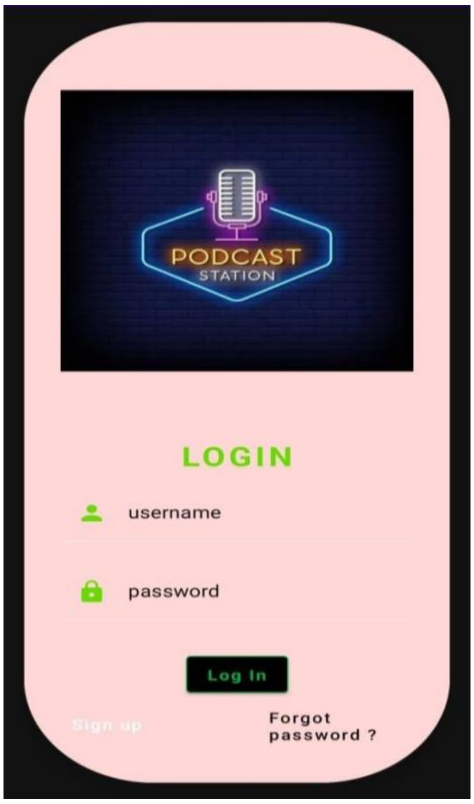
REGISTRATIONACTIVITY.KT

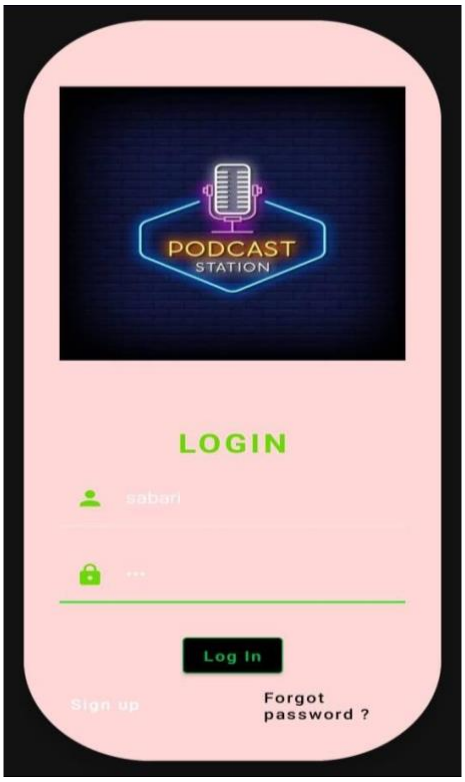
```
package com.example.podcastplayer


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.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.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 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("") }
val hue = 193f / 360f         // Convert hue to a fraction of 360
val saturation = 1.0f         // Saturation as a decimal (100%)
val lightness = 0.92f         // Lightness as a decimal (92%)



Column(
    Modifier
        .background(Color.hsl(hue,saturation,lightness))
        .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
        )
    }



    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)
}
```

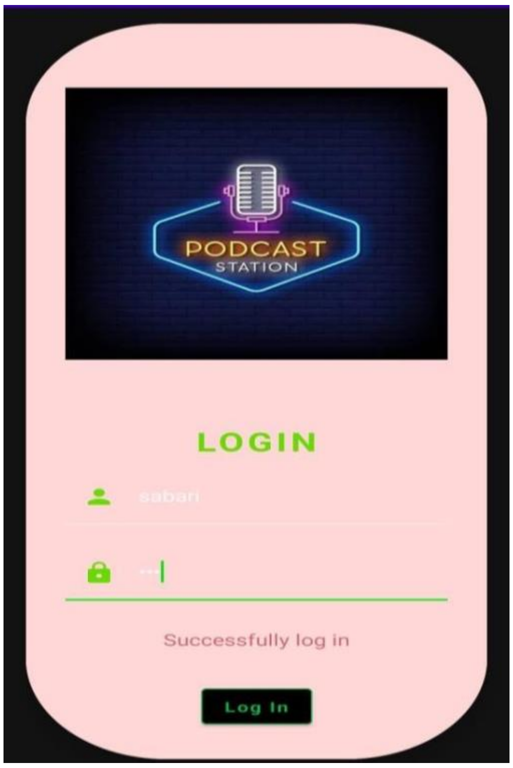
SCREENSHOTS





sabari

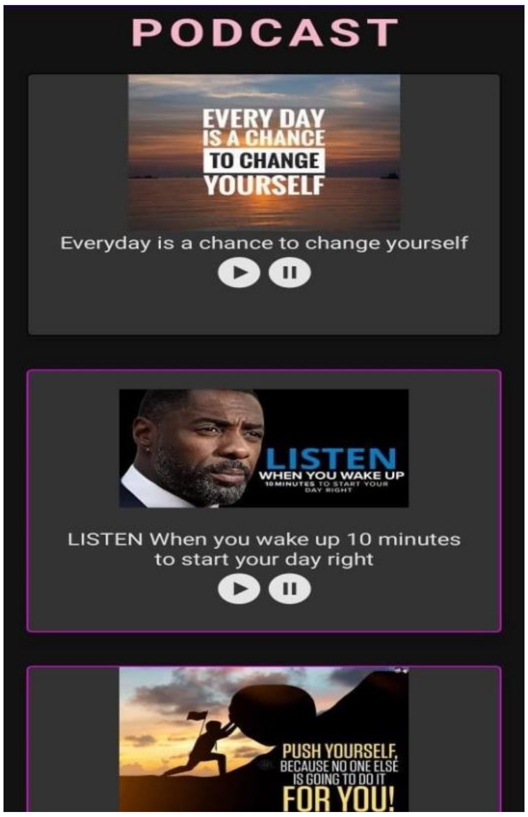
•••



LOGIN

sabari

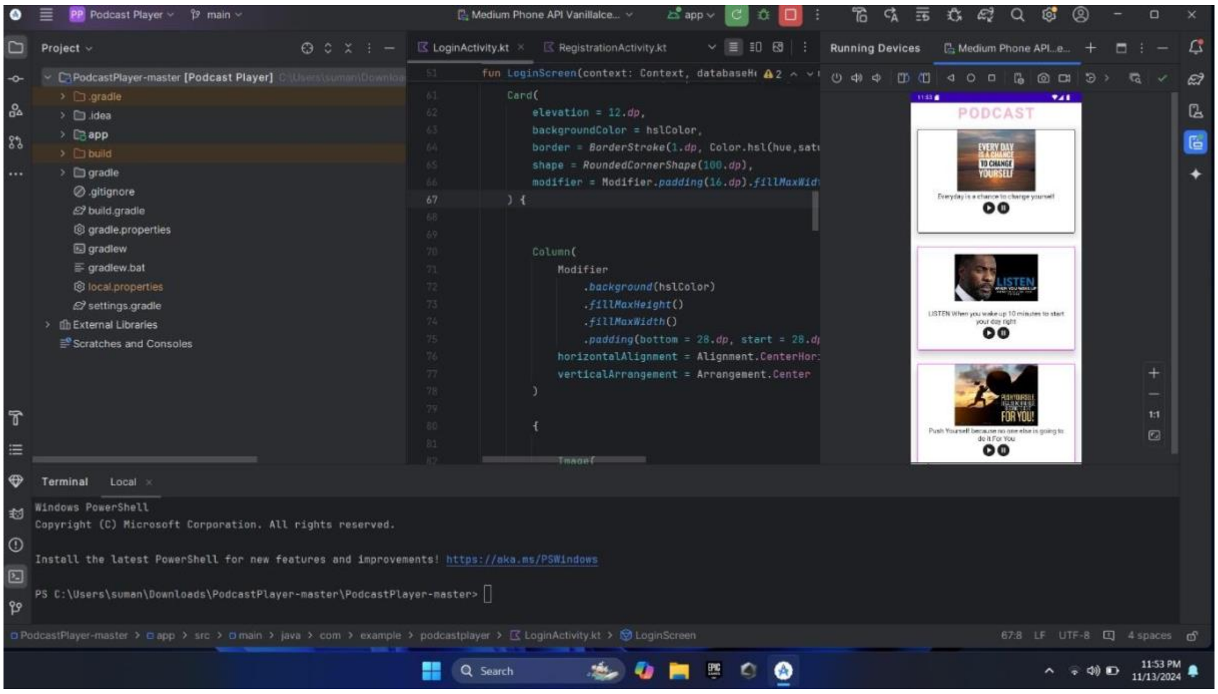
Successfully log in


EVERY DAY IS A CHANCE TO CHANGE YOURSELF
LISTEN WHEN YOU WAKE UP 10 MINUTES TO START YOUR DAY RIGHT
PUSH YOURSELF, BECAUSE NO ONE ELSE IS GOING TO DO IT FOR YOU!

PODCAST

Everyday is a chance to change yourself

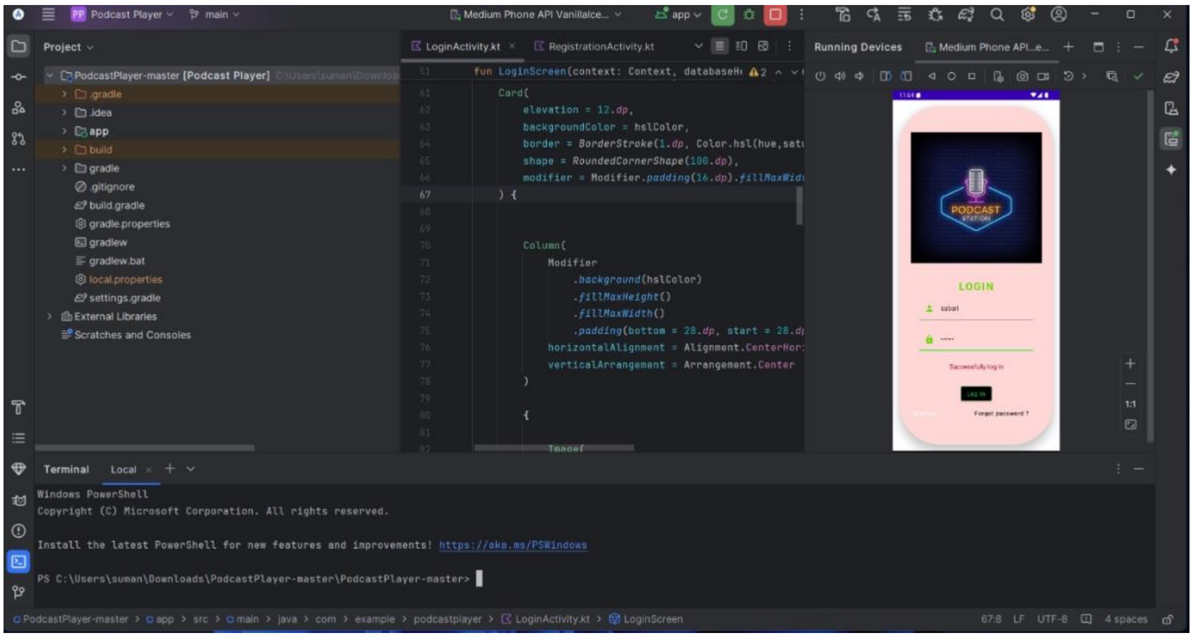
LISTEN When you wake up 10 minutes to start your day right



```
fun LoginScreen(context: Context, databaseH
        Card(
            elevation = 12.dp,
            backgroundColor = hslColor,
            border = BorderStroke(1.dp, Color.hsl(hue,satu
            shape = RoundedCornerShape(100.dp),
            modifier = Modifier.padding(16.dp).fillMaxWid
        ) {


            Column(
                Modifier
                    .background(hslColor)
                    .fillMaxHeight()
                    .fillMaxWidth()
                    .padding(bottom = 28.dp, start = 28.dp
                horizontalAlignment = Alignment.CenterHor
                verticalArrangement = Arrangement.Center
            )

            {

                Image(
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\suman\Downloads\PodcastPlayer-master\PodcastPlayer-master>
```

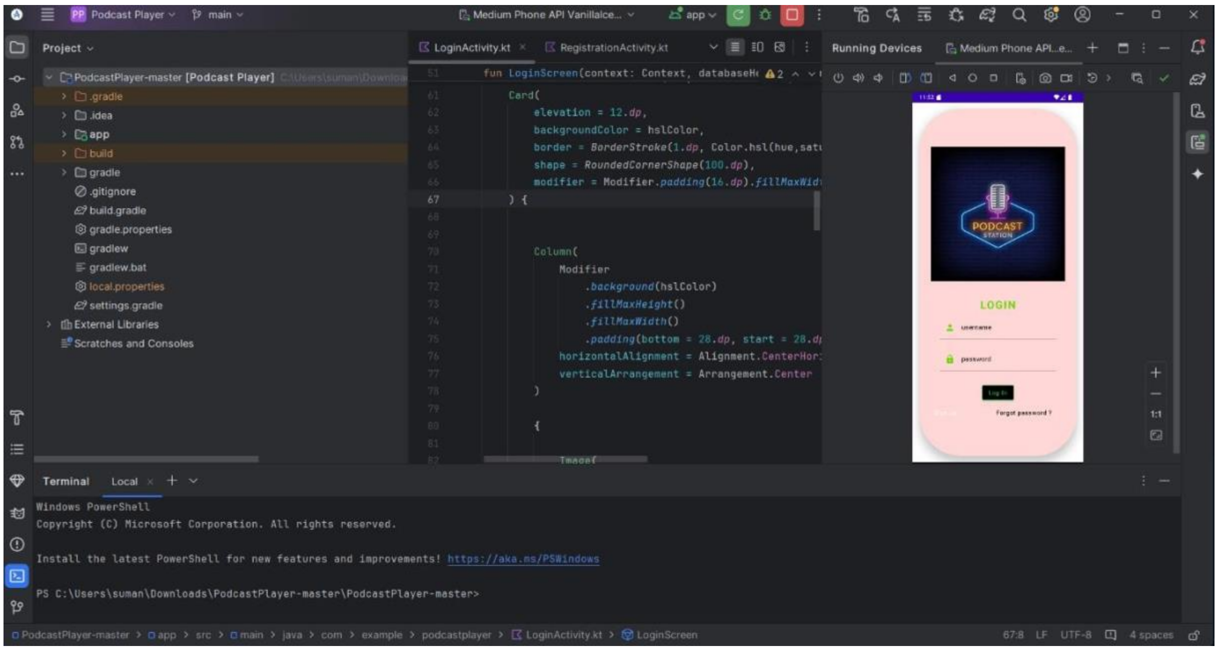
PodcastPlayer-master > app > src > main > java > com > example > podcastplayer > LoginActivity.kt > LoginScreen   67:8 LF UTF-8 4 spaces



```
fun LoginScreen(context: Context, databaseH
        Card(
            elevation = 12.dp,
            backgroundColor = hslColor,
            border = BorderStroke(1.dp, Color.hsl(hue,satu
            shape = RoundedCornerShape(100.dp),
            modifier = Modifier.padding(16.dp).fillMaxWid
        ) {


            Column(
                Modifier
                    .background(hslColor)
                    .fillMaxHeight()
                    .fillMaxWidth()
                    .padding(bottom = 28.dp, start = 28.dp
                horizontalAlignment = Alignment.CenterHor
                verticalArrangement = Arrangement.Center
            )

            {

                Image(
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\suman\Downloads\PodcastPlayer-master\PodcastPlayer-master>
```

PodcastPlayer-master > app > src > main > java > com > example > podcastplayer > LoginActivity.kt > LoginScreen   67:8 LF UTF-8 4 spaces



```
fun LoginScreen(context: Context, databaseH
    Card(
        elevation = 12.dp,
        backgroundColor = hslColor,
        border = BorderStroke(1.dp, Color.hsl(hue,satu
        shape = RoundedCornerShape(100.dp),
        modifier = Modifier.padding(16.dp).fillMaxWid
    ) {


        Column(
            Modifier
                .background(hslColor)
                .fillMaxHeight()
                .fillMaxWidth()
                .padding(bottom = 28.dp, start = 28.dp
            horizontalAlignment = Alignment.CenterHor
            verticalArrangement = Arrangement.Center
        )

        {

            Image(
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\suman\Downloads\PodcastPlayer-master\PodcastPlayer-master>
```

DEMO VIDEO :

CONCLUSION

The development of Podcast Plus showcased the effective use of modern Android development practices, leveraging Kotlin for its efficient and scalable coding capabilities. By integrating a Redux-inspired state management pattern and implementing dynamic theming, the app offers a highly customizable and engaging user experience. Throughout the development process, the team effectively navigated technical challenges such as ensuring smooth state transitions and optimizing performance across a variety of devices and user preferences. This project demonstrated the team’s ability to deliver a feature-rich, intuitive, and scalable app that not only meets but exceeds user expectations in the podcast space.As the app evolves, future enhancements will focus on expanding functionality, improving content discovery, and incorporating more advanced features such as personalized recommendations and integration with external platforms. Continued emphasis will be placed on user satisfaction, ensuring that Podcast Plus remains a top-tier platform in the competitive podcast app landscape. By maintaining a user-centric approach and adopting best practices in development and testing, Podcast Plus is poised for long-term success.