APP WORLD

ANIMAÇÃO E MULTIMÍDIA

13A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 ANIMAÇÃO E MULTIMÍDIA

1.1 Aplicando efeitos de animação

A biblioteca de animações do Jetpack Compose fornece uma série de animações que tornam a aplicação muito mais atraente ao usuário. Com ela, nós podemos aplicar movimento e transições mais fluídas entre os diferentes componentes da tela. Temos basicamente 4 grupos de animação:

Fade: que aplica um efeito de esmaecimento na entrada e saída do componente;

Slide: aplica efeito de deslizamento na entrada e saída do componente. O deslizamento pode ser aplicado vertical ou horizontalmente;

Scale: aplica uma alteração do tamanho do componente no momento de entrada ou saída da tela;

Expand: usado para aplicar um efeito de expansão no tamanho do componente.

Todos esses efeitos podem ser configurados para diferentes comportamentos, inclusive é possível juntar um ou mais efeitos para criarmos um efeito personalizado.

Para testarmos os diferentes tipos de animação que podemos incluir em nossos componentes, crie um projeto no Android Studio com o nome “Animacao”. Substitua as funções “Greeting” e “GreetingPreview” pelas funções disponíveis na listagem “Layout inicial da aplicação”. Não se esqueça de chamar a função “BoxScreen” no método “onCreate” da classe “MainActivity”.

```
package br.com.fiap.animacao

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.animation.EnterTransition
import androidx.compose.animation.ExitTransition
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
```

```
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import br.com.fiap.animacao.ui.theme.AnimacaoTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      AnimacaoTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          AnimacaoScreen()
        }
      }
    }
  }
}

@OptIn(ExperimentalAnimationApi::class)
@Composable
fun AnimacaoScreen() {
  var visible = remember {
    mutableStateOf(false)
  }
  var enter = remember {
    mutableStateOf(fadeIn())
  }
  var exit = remember {
    mutableStateOf(fadeOut())
  }

  Column(
    horizontalAlignment = Alignment.CenterHorizontally,
    modifier = Modifier
      .fillMaxWidth()
      .padding(16.dp)
  ) {
    Row(
      verticalAlignment = Alignment.CenterVertically,
      horizontalArrangement = Arrangement.SpaceBetween,
      modifier = Modifier.fillMaxWidth()
    ) {
      Button(onClick = {
```

```
        }) {
          Text(text = "Fade")
        }
        Button(onClick = {
        }) {
          Text(text = "Slide")
        }
        Button(onClick = {
        }) {
          Text(text = "Scale")
        }
        Button(onClick = {
        }) {
          Text(text = "Expand")
        }
      }
      Spacer(modifier = Modifier.height(64.dp))
      BoxComponent(
        visible = visible.value,
        enter = enter.value,
        exit = exit.value
      )
    }
}

@Composable
fun BoxComponent(
  visible: Boolean,
  enter: EnterTransition,
  exit: ExitTransition
) {
  AnimatedVisibility(
    visible = visible,
    enter = enter,
    exit = exit
  ) {
    Box(modifier = Modifier
      .size(200.dp)
      .background(color = Color.Red))
  }
}
```

Código-fonte 1 – Layout inicial da aplicação
Fonte: Elaborado pelo autor (2023)

A função “BoxScreen” é responsável por renderizar a tela enquanto a função “BoxComponent” é responsável por renderizar a “Box” que utilizaremos para testar os efeitos de animação.

A função “BoxComponent” recebe três argumentos:

- **visible**: argumento do tipo booleano que é responsável por controlar a visibilidade do componente na tela;
- **enter**: argumento do tipo “EnterTransition” que é responsável pelo efeito aplicado durante a entrada do componente “Box” na tela;

- **exit**: argumento do tipo “ExitTransition” que é responsável pelo efeito aplicado durante a saída do componente “Box” da tela.

A função “AnimacaoScreen” declara e inicia três variáveis de estado que são responsáveis por armazenar o estado de exibição e animação de entrada ou saída do componente. Essa função implementa 4 botões que são responsáveis por aplicar os diferentes tipos de animação que vamos testar.

Ao executar a aplicação, a interface do usuário deverá se parecer com a figura “Layout inicial da aplicação”:

Figura 1 – Layout inicial da aplicação
Fonte: Elaborado pelo autor (2023)

1.1.1 Implementando o efeito de FadeIn/FadeOut

O efeito de esmaecimento do componente é chamado de “Fade”. Vamos aplicar o efeito de esmaecimento do componente “Box” durante a entrada e saída. O botão “Fade” deverá se parecer com a listagem “Aplicando efeito de esmaecimento”, logo abaixo:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = fadeIn()
  exit.value = fadeOut()
}) {
  Text(text = "Fade")
}
```

Código-fonte 2 – Aplicação de efeito de esmaecimento
Fonte: Elaborado pelo autor (2023)

Ao clicar no botão “Fade”, atualizamos o valor da variável de estado “visible” para “true”. Assim, o componente “Box” será exibido com efeito de esmaecimento “fadeIn”. Ao clicar novamente, a variável será atualizada para “false”, aplicando o efeito de saída “fadeOut". Experimente!

Vamos alterar o tempo em que as animações ocorrem. Altere o seu código de acordo com a listagem “Alterando tempo de esmaecimento”:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = fadeIn(animationSpec = tween(5000))
  exit.value = fadeOut(animationSpec = tween(5000))
}) {
  Text(text = "Fade")
}
```

Código-fonte 3 – Alterando tempo de esmaecimento
Fonte: Elaborado pelo autor (2023)

Ao clicar no botão “Fade” o tempo em que o efeito de esmaecimento tanto na entrada quanto na entrada levará será de 5 mil milissegundos, ou seja, 5 segundos.

1.1.2 Aplicando efeito de deslizamento

Para aplicarmos um efeito de deslizamento ao componente, vamos utilizar a fução “slide”. Essa função possui as seguintes variações slideInHorizontally, slideOutHorizontally, slideInVertically e slideOutVertically, que são autoexplicativos. O botão “Slide” será implementado de acordo com a listagem “Aplicando o efeito de deslizamento”:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = slideInHorizontally()
  exit.value = slideOutHorizontally()
}) {
  Text(text = "Slide")
}
```

Código-fonte 4 – Aplicando o efeito de deslizamento
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e observe o efeito de deslizamento na entrada e saída do componente.

Troque o efeito de saída para “slideOutVertically” e observe o efeito. Muito legal né? É possível juntarmos efeitos! Vamos colocar um efeito de “fadeOut” junto com o efeito de “slideOutVertically”. O código deverá se parecer com a listagem “Deslizamento e esmaecimento juntos”:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = slideInHorizontally()
  exit.value = slideOutVertically() + fadeOut(animationSpec =
tween(2000))
}) {
  Text(text = "Slide")
}
```

Código-fonte 5 – Deslizamento e esmaecimento juntos
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e experimente o resultado.

1.1.3 Aplicando efeito de escala

O efeito de escala aplica um efeito de alteração de tamanho do componente durante a entrada ou saída da tela. O uso das funções “scaleIn” e “scaleOut” pode ser visualizado na listagem “Aplicando o efeito de escala”. O código deve ser escrito no botão “Scale”.

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = scaleIn()
  exit.value = scaleOut()
}) {
  Text(text = "Scale")
}
```

Código-fonte 6 – Aplicando o efeito de escala
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e teste o efeito de escala. Percebeu que a escala ocorre a partir do centro do componente? Também é possível mudar o tempo da animação assim como combinar com outras animações. Experimente!

1.1.4 Aplicando efeito de expansão

Assim como o efeito de escala, o efeito de expansão ocorre no tamanho da imagem, mas começando pelas laterais horizontal ou verticalmente. A aplicação do efeito de expansão pode ser obtida na listagem “Aplicando o efeito de expansão”:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = expandHorizontally()
  exit.value = shrinkHorizontally()
}) {
  Text(text = "Expand")
}
```

Código-fonte 7 – Aplicando o efeito de expansão
Fonte: Elaborado pelo autor (2023)

Execute a aplicação e observe o efeito.

Neste exemplo, você notou que o efeito ocorre na horizontal. Também é possível alterarmos o efeito para vertical. O código deve se parecer como na listagem “Expansão na vertical”:

```
Button(onClick = {
  visible.value = !visible.value
  enter.value = expandVertically()
  exit.value = shrinkVertically()
}) {
  Text(text = "Expand")
}
```

Código-fonte 8 – Expansão na vertical
Fonte: Elaborado pelo autor (2023)

Vale a pena lembrar que também é possível alternar os valores de entrada e saída, além de juntar com outros efeitos. Use a sua criatividade e crie a sua animação.

2 MULTIMÍDIA

É muito comum a utilização de recursos áudio visuais em aplicativos Android. É bastante útil emitir um som para avisar o usuário sobre algum estado que mudou, ou uma mensagem que chegou. Além disso, a utilização de vídeo também é bastante comum nos apps.

2.1 Reproduzindo áudio

Crie um projeto no Android Studio com o nome “Audio Player”. Remova as funções “Greeting” e “GreetingPreview”. Não se esqueça de remover a chamada para a função “Greeting” do método “onCreate” da classe “MainActivity”.

Para aprendermos como reproduzir áudio em aplicativos Android será necessário disponibilizarmos um áudio como recurso para a nossa aplicação. Então, vamos criar uma nova pasta de recursos na pasta “res” do nosso projeto.

Clique com o botão direito do mouse na pasta “res”, aponte para “New” e clique em “Android Resource Directory”, conforme mostra a figura “Criando a pasta raw”:

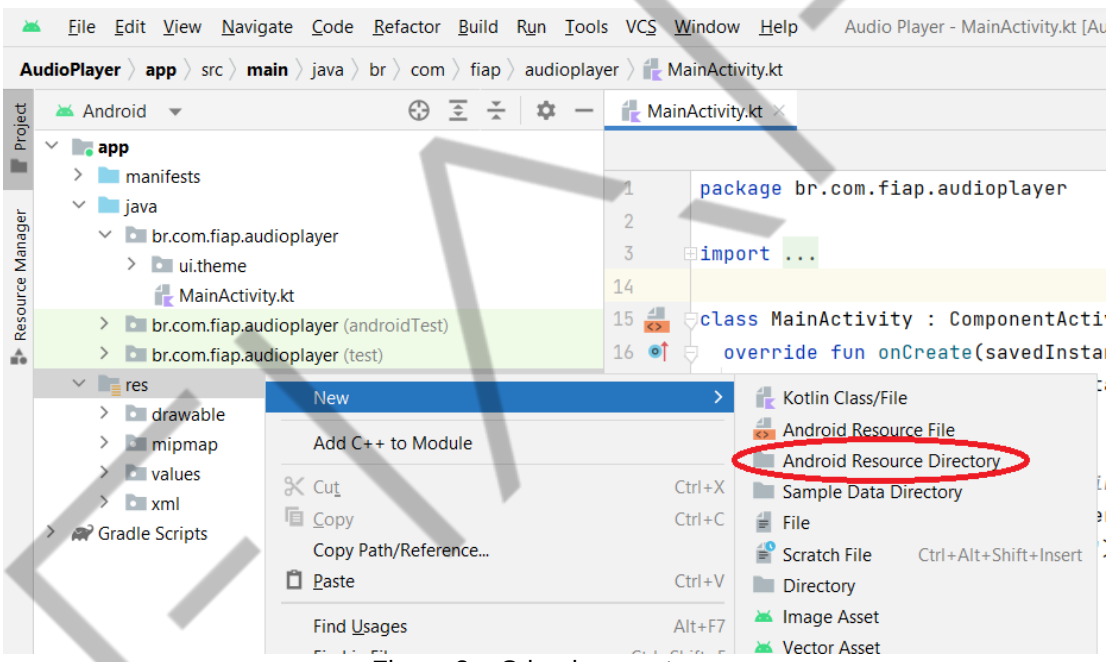

Figura 2 – Criando a pasta raw
Fonte: Elaborado pelo autor (2023)

Na tela “New Resource Directory” selecione a opção “raw” no campo “Resource Type”, conforme a figura “New Resource Directory” e clique em “OK”.

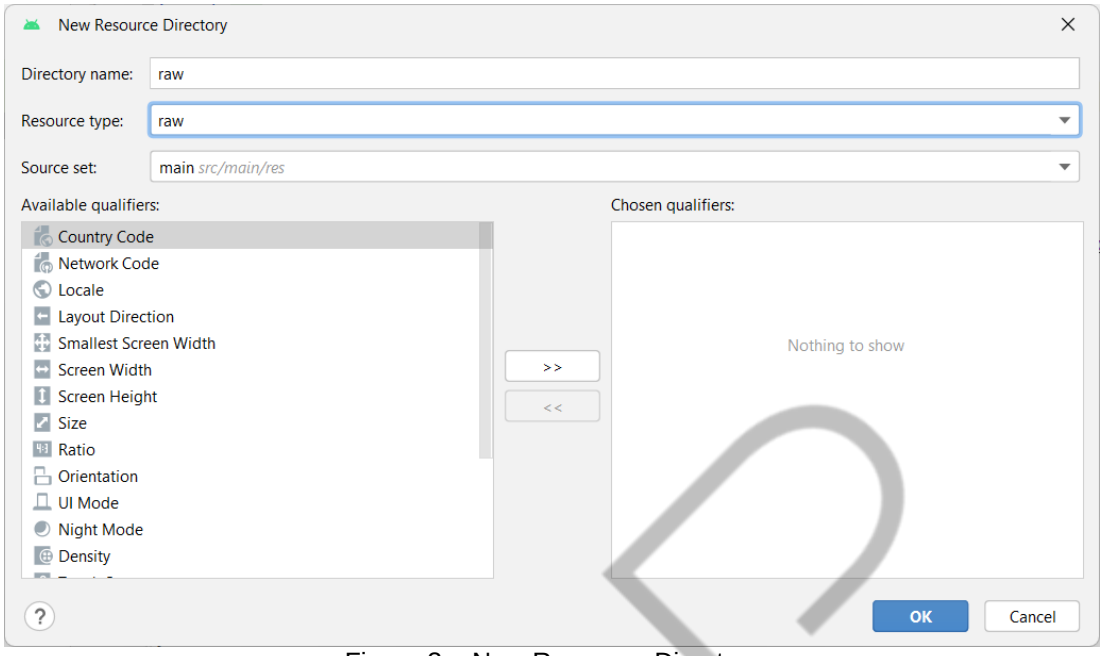

Figura 3 – New Resource Directory
Fonte: Elaborado pelo autor (2023)

Guarde o arquivo de áudio que você utilizará na pasta “raw” que acabamos de criar. A estrutura do projeto deverá se parecer com a figura “Estrutura de arquivos do projeto”:

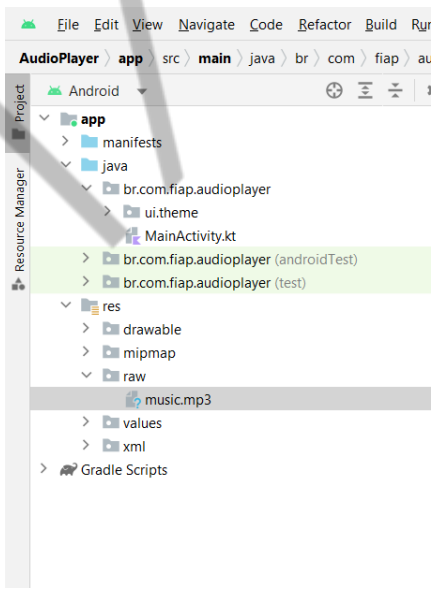

Figura 4 – Estrutura de arquivos do projeto
Fonte: Elaborado pelo autor (2023)

Agora, coloque o código da listagem “Layout inicial do projeto” no arquivo “MainActivity.kt”.

```
@Composable
fun AudioPlayer(context: Context) {

  var player = remember {
```

```
    mutableStateOf(MediaPlayer.create(context, R.raw.music))
  }

  Box(contentAlignment = Alignment.Center) {
    Row() {

      IconButton(onClick = { }) {
        Icon(
          painter = painterResource(id = R.drawable.play),
          contentDescription = ""
        )
      }

      IconButton(onClick = { }) {
        Icon(
          painter = painterResource(id = R.drawable.pause),
          contentDescription = ""
        )
      }

      IconButton(onClick = { }) {
        Icon(
          painter = painterResource(id = R.drawable.stop),
          contentDescription = ""
        )
      }

    }
  }
}
```

Código-fonte 9 – Layout inicial do projeto
Fonte: Elaborado pelo autor (2023)

Os arquivos de imagem utilizados como botões do nosso tocador de música deverão ser colocados na pasta “res/drawable”. Use as imagens da sua preferência. A estrutura de pastas do projeto deverá se parecer com a figura “Arquivos de imagem do projeto”:

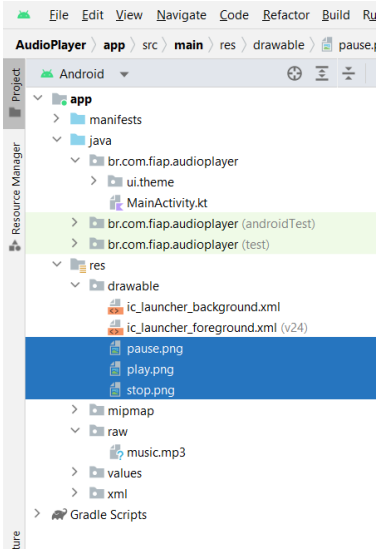

Figura 5 – Arquivos de imagem do projeto
Fonte: Elaborado pelo autor (2023)

Antes de executar o projeto, modifique o método “onCreate” da classe “MainActivity” para chamar a função “AudioPlayer, conforme a listagem de código “Chamando a função AudioPlayer”:

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      AudioPlayerTheme {
        Surface(modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background) {
          AudioPlayer(this)
        }
      }
    }
  }
}
```

Código-fonte 10 – Chamando a função AudioPlayer
Fonte: Elaborado pelo autor (2023)

Ao executar o aplicativo, o layout inicial deverá se parecer com a figura “Layout da aplicação”:

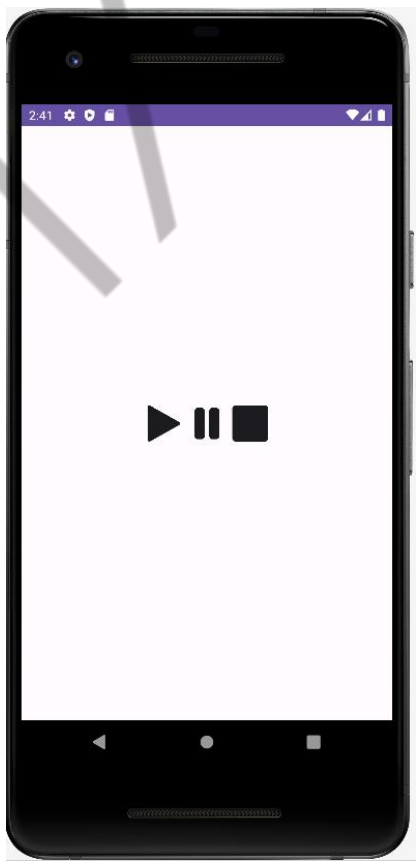

Figura 6 – Layout da aplicação
Fonte: Elaborado pelo autor (2023)

Para a reprodução do áudio, vamos utilizar a classe “MediaPlayer”, que nos fornece todos os métodos necessários para tocar, pausar, parar etc. No início da função “AudioPlayer” crie uma variável de estado para guardar a referência para o objeto “MediaPlayer”. Seu código deverá se parecer com a listagem “Criação da variável de estado player”, logo abaixo:

```
@Composable
fun AudioPlayer(context: Context) {

  var player = remember {
    mutableStateOf(MediaPlayer.create(context, R.raw.music))
  }

. . . trecho de código omitido
```

Código-fonte 11 – Criação da variável de estado player
Fonte: Elaborado pelo autor (2023)

A implementação do botão “Play” pode ser obtida na listagem “Implementação do botão Play”, abaixo:

```
IconButton(onClick = {
  if (player.value == null){
    player.value = MediaPlayer.create(context, R.raw.music)
  }
  player.value.start()
}) {
  Icon(
    painter = painterResource(id = R.drawable.play),
    contentDescription = ""
  )
}
```

Código-fonte 12 – Implementação do botão Play
Fonte: Elaborado pelo autor (2023)

Antes de chamarmos o método “start()” verificamos se a variável “player” não está nula; se estiver, instanciamos novamente o objeto “MediaPlayer” e em seguida tocamos o áudio. Execute a aplicação e ouça a sua música.

A implementação do botão “Pause” é bastante simples. O código deverá se parecer com a listagem “Implementação do botão Pause”:

```
IconButton(onClick = {
  player.value.pause()
}) {
  Icon(
    painter = painterResource(id = R.drawable.pause),
    contentDescription = ""
  )
}
```

Código-fonte 13 – Implementação do botão Pause
Fonte: Elaborado pelo autor (2023)

Execute a aplicação, pressione “play” e em seguida pressione “pause”. A música deverá ser pausada até que “play” seja pressionada novamente.

O botão “stop”, também é bastante simples. O código do botão “stop” pode ser consultado na listagem “Implementação do botão Stop”:

```
IconButton(onClick = {
  player.value.stop()
  player.value.reset()
  player.value.release()
  player.value = null
}) {
  Icon(
    painter = painterResource(id = R.drawable.stop),
    contentDescription = ""
  )
}
```

Código-fonte 14 – Implementação do botão Stop
Fonte: Elaborado pelo autor (2023)

Quando pressionamos o botão “stop”, precisamos garantir que a variável player ficará totalmente “limpa”, por isso utilizamos os métodos “reset()” e “release()”, além de atualizarmos o valor para “null”.

Execute a aplicação e teste as funcionalidades dos três botões. Agora você já pode ouvir a sua música preferida em um aplicativo desenvolvido por você.

2.2 Reproduzindo vídeos

A reprodução de vídeo no Android também é bastante simples. Vamos criar um projeto no Android Studio com o nome “Video Player”. Assim que o projeto estiver concluído vamos adicionar uma biblioteca externa chamada “Exoplayer”. Então, abra o arquivo “build.gradle (Module: app)” e adicione no bloco “dependencies” a instrução de importação conforme a listagem “Adicionando dependência do Exoplayer”, logo abaixo:

```
dependencies {

    implementation 'androidx.core:core-ktx:1.8.0'
    implementation platform('org.jetbrains.kotlin:kotlin-bom:1.8.0')
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3'
```

```
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-
core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'

    // Exoplayer
    implementation "com.google.android.exoplayer:exoplayer:2.19.0"
}
```

Código-fonte 15 – Adicionando a dependência do Exoplayer
Fonte: Elaborado pelo autor (2023)

Após a inclusão da linha acima sincronize o Gradle para que o “Exoplayer” seja implementado no projeto.

Apague as funções “Greeting” e “GreetingPreview”, e adicione a função “VideoPlayer” conforme a listagem de código “Função VideoPlayer”:

```
@Composable
fun VideoPlayer() {
  val videoUrl =
    "https://commondatastorage.googleapis.com/gtv-videos-
bucket/sample/BigBuckBunny.mp4"
  val context = LocalContext.current
  val player = ExoPlayer.Builder(context).build()
  val playerView = PlayerView(context)
  val mediaItem = MediaItem.fromUri(videoUrl)

  val playWhenReay by remember {
    mutableStateOf(true)
  }

  player.setMediaItem(mediaItem)
  playerView.player = player

  LaunchedEffect(player) {
    player.prepare()
    player.playWhenReady = playWhenReay
  }

  AndroidView(factory = {
    playerView
  })

}
```

Código-fonte 16 – Função VideoPlayer
Fonte: Elaborado pelo autor (2023)

Na função “VideoPlayer”, criamos os seguintes itens:

- **videoUrl**: variável que armazena o caminho para o vídeo que queremos reproduzir;

- **player**: responsável por reproduzir o vídeo;
- **playerView**: responsável por exibir o vídeo e seus controles;
- **mediaItem**: o objeto de mídia que será reproduzido;
- **playWhenReady**: um booleano que indica que o conteúdo já está disponível para ser reproduzido;
- **LaunchedEffect**: inicia a corrotina responsável pela reprodução do vídeo;
- **AndroidView**: responsável por colocar o componente “playerView” na hierarquia de exibição na interface do usuário.

Execute a aplicação em um emulador. De acordo com a figura “Reproduzindo vídeos no Android”, você será capaz de visualizar o vídeo e interagir com ele.

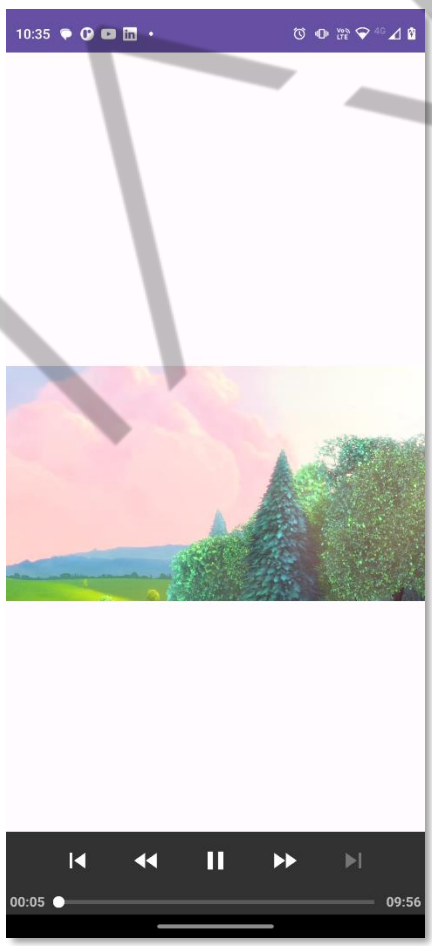

Figura 7 – Reproduzindo vídeos no Android
Fonte: Elaborado pelo autor (2023)

3 PUBLICANDO A APLICAÇÃO NO GOOGLE PLAY

Agora chegou a hora de distribuir a sua aplicação para o mundo e isso é feito a partir da publicação da aplicação na loja do Google conhecida como Google Play.

3.1 Criação de conta no Google Play Console

Para publicar aplicações devemos criar uma conta de desenvolvedor. Para esse cadastro é necessário efetuarmos o pagamento de US$25 dólares. A boa notícia é que esse será o único momento que você pagará alguma coisa, já que com o pagamento dessa taxa inicial você poderá publicar quantos aplicativos quiser. O cadastro de uma conta de desenvolvedor deve ser feito no console do Google Play (https://play.google.com/console/about/). Assim que acessar o caminho, clique no link “Go to Play Console” no canto superior direito da tela, conforme a figura “Google Play Console”. Se você não tiver uma conta de desenvolvedor você será convidado a cria-la. O processo é bastante simples.

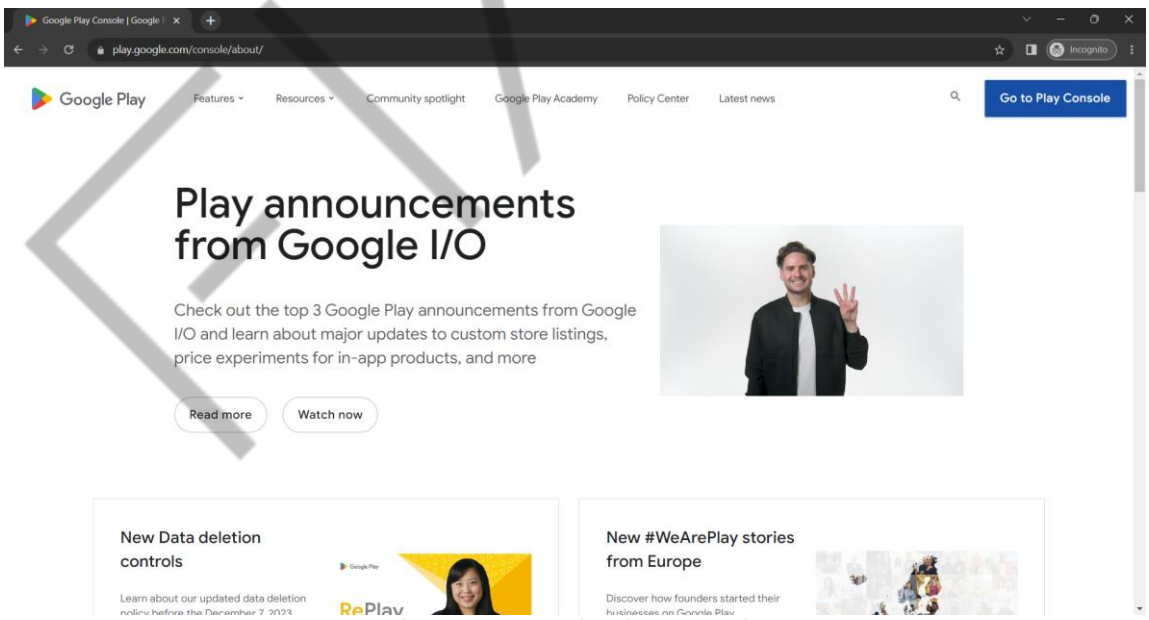

Figura 8 – Google Play Console
Fonte: Elaborado pelo autor (2023)

3.2 Preparação do aplicativo para publicação

Para publicarmos a aplicação, precisamos gerar um pacote do tipo “aab” – “Android App Bundle”, que é um formato de publicação que inclui todos os recursos e código compilados do app. O formato “aab” é o novo padrão de empacotamento de aplicativos para publicação utilizados pelo Google. Para criar um pacote “aab” abra o projeto que deseja publicar e clique no menu “Build”, e em seguida clique na opção “Generate Signed Bundle / APK ...”, conforme a figura “Generate Signed Bundle”:

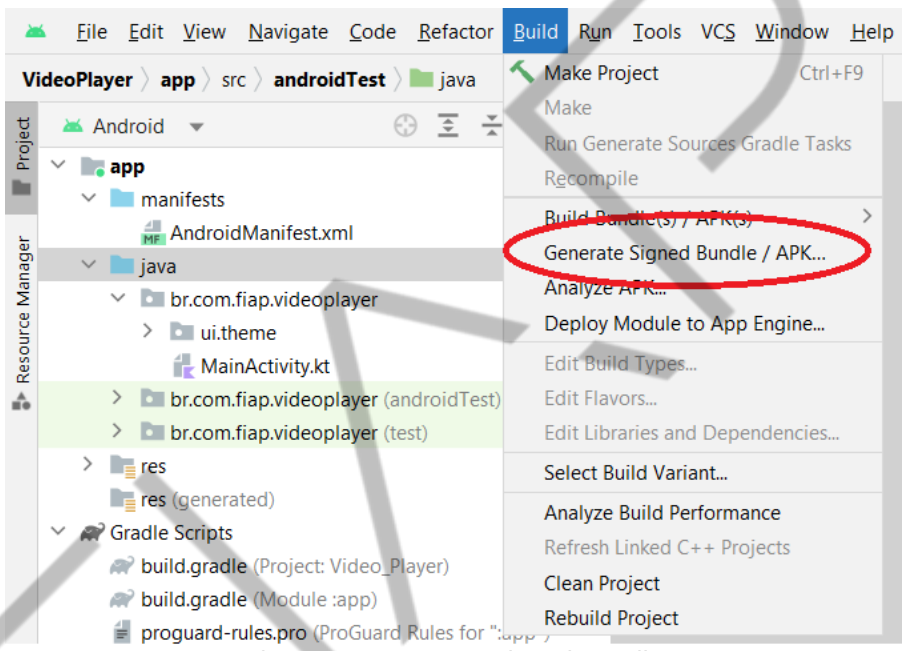

Figura 9 – Generate Signed Bundle
Fonte: Elaborado pelo autor (2023)

Na janela “Generate Signed Bundle or APK” mantenha a opção “Android App Bundle” selecionada e pressione “Next”, conforme a figura “Janela Generate Signed Bundle”:

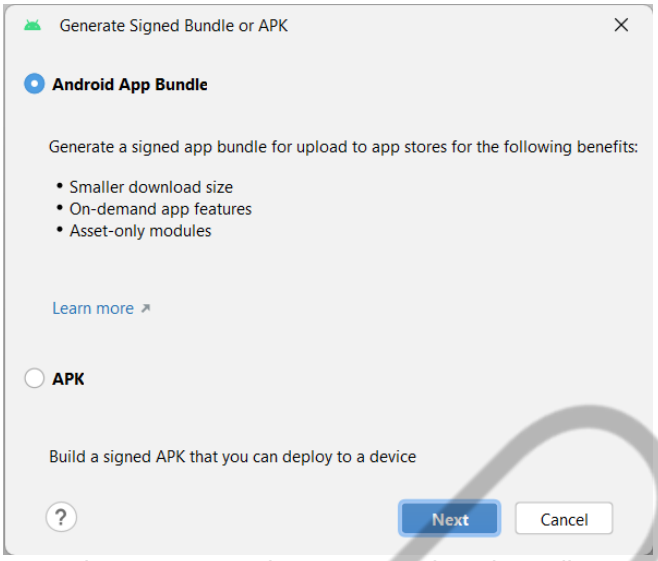

Figura 10 – Janela Generate Signed Bundle
Fonte: Elaborado pelo autor (2023)

Em seguida, na próxima janela, clique no botão “Create new...”, conforme a figura “Criar nova chave”:

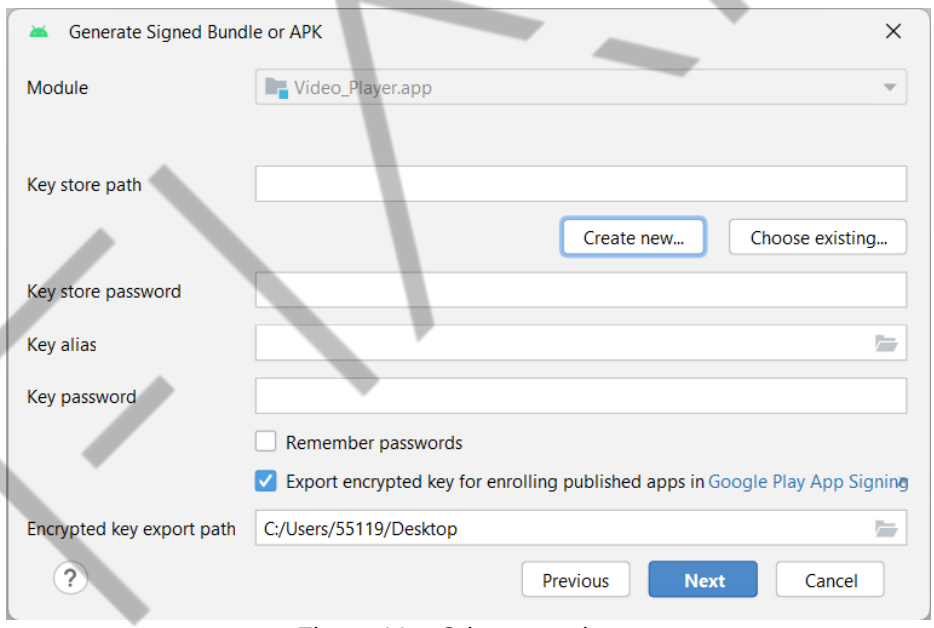

Figura 11 – Criar nova chave
Fonte: Elaborado pelo autor (2023)

Na janela “New Key Store”, clique no ícone de pasta no campo “Key store path”, conforme a figura “New Key Store” e selecione o local e nome da chave, conforme a figura “Choose keystore file” e clique no botão “OK”.

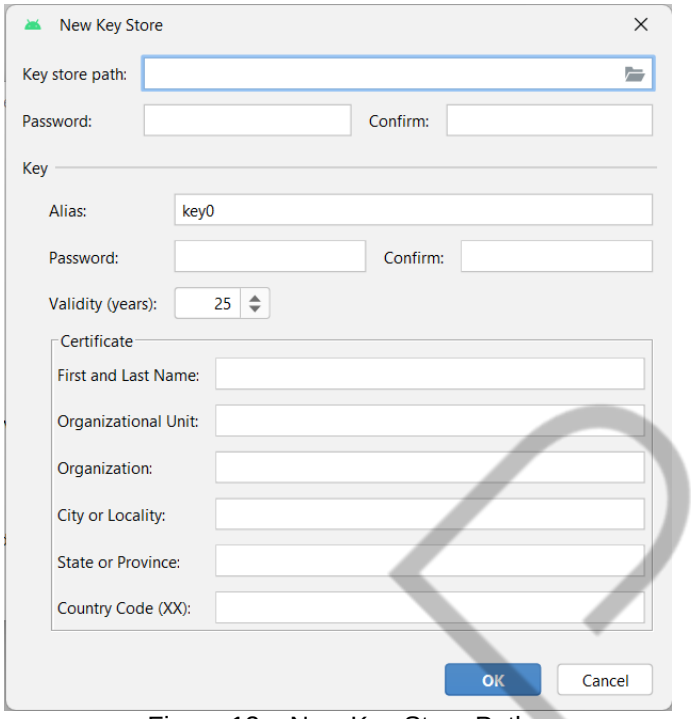

Figura 12 – New Key Store Path
Fonte: Elaborado pelo autor (2023)

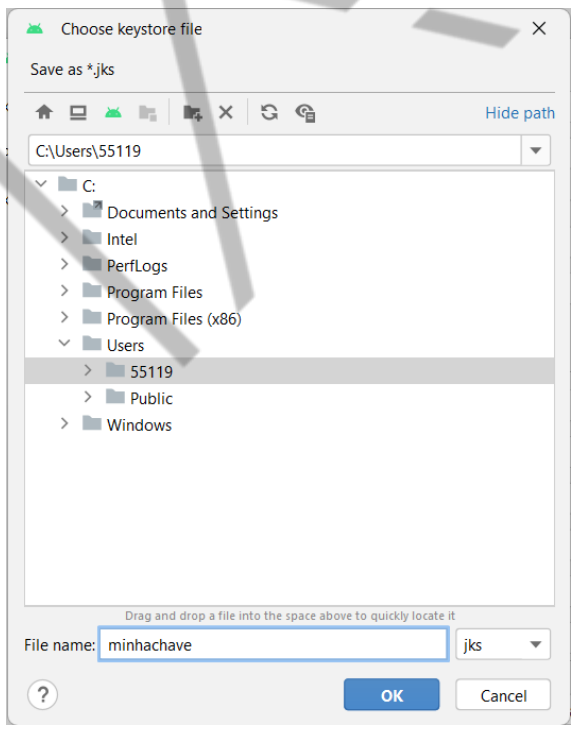

Figura 13 – Choose keystore file
Fonte: Elaborado pelo autor (2023)

Ao retornar para a janela “New Key Store” preencha os seus dados no formulário para geração do certificado digital, conforme a figura “Dados do certificado” e clique no botão “OK”.

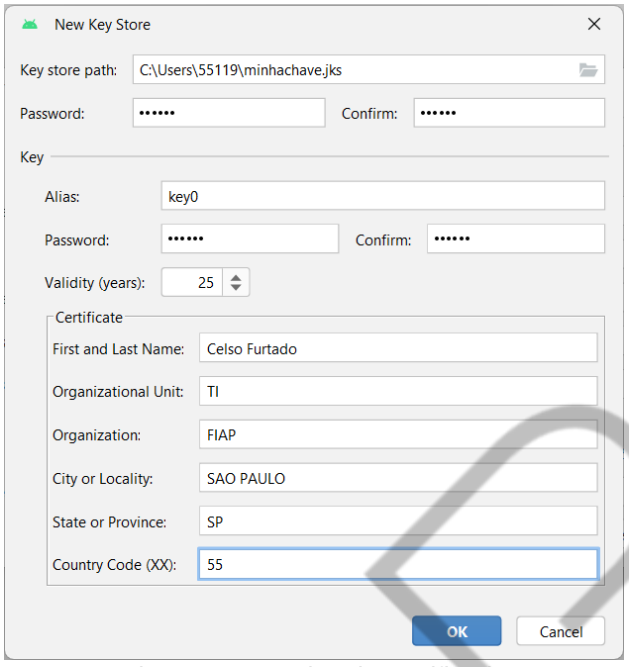

Figura 14 – Dados do certificado
Fonte: Elaborado pelo autor (2023)

Ao retornar para a janela “Generate Signed Bundle or APK”, clique em “Next”, conforme a figura “Dados da chave”:

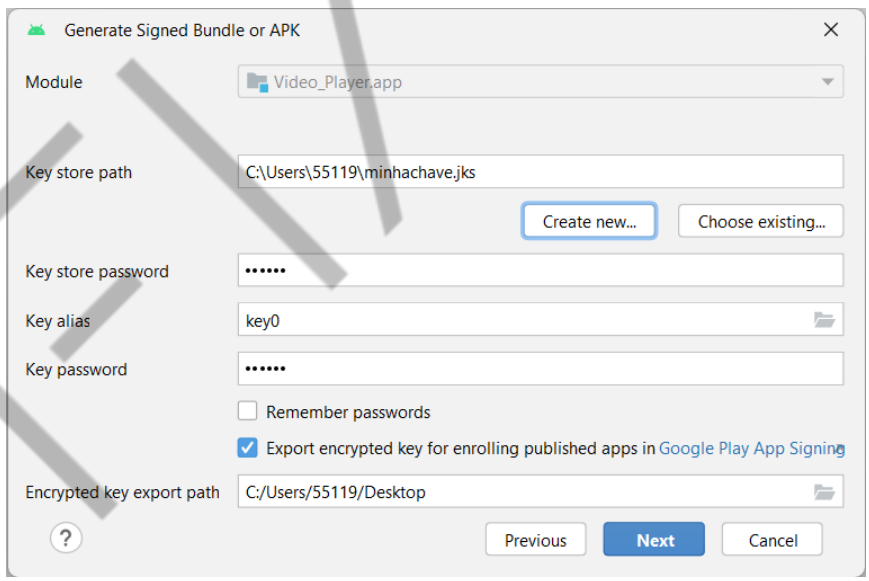

Figura 15 – Dados da chave
Fonte: Elaborado pelo autor (2023)

Na janela seguinte selecione “release” e pressione o botão “Create”, conforme a figura “Destination Folder”:

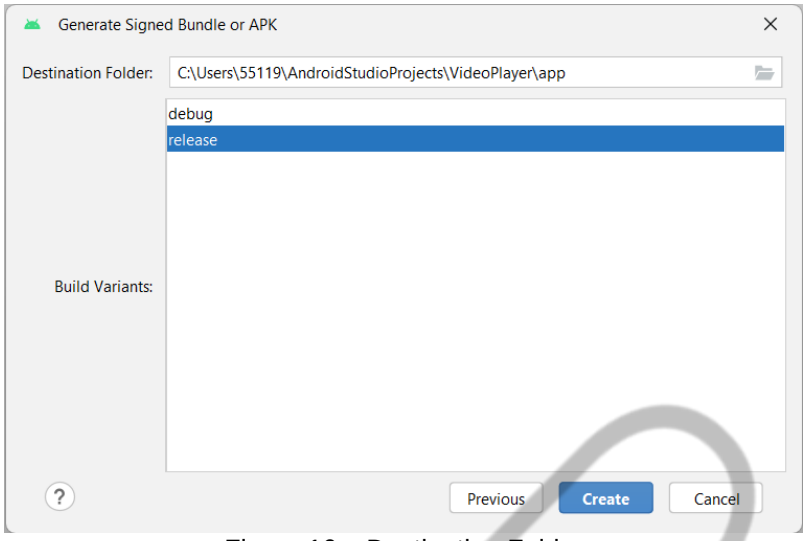

Figura 16 – Destination Folder
Fonte: Elaborado pelo autor (2023)

Abra a pasta indicada no campo “Destination Folder” e verifique se o arquivo “aab” foi gerado, conforme a figura “Arquivo aab”.

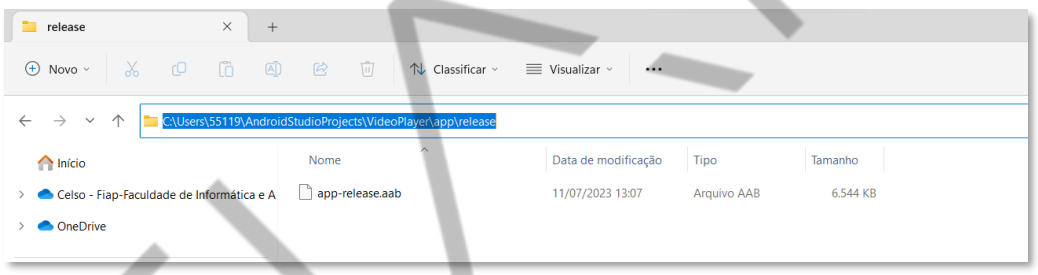

Figura 17 – Arquivo aab
Fonte: Elaborado pelo autor (2023)

3.3 Criando o App no Google Play

Após a geração do arquivo “aab” vamos criar um app no Google Play, portanto acesse o Google Play Console com a sua conta de desenvolvedor e clique no botão “Criar App” no canto superior direito da tela, conforme a figura “Criar App no Google Play”:

Figura 18 – Criar App no Google Play
Fonte: Elaborado pelo autor (2023)

Na janela “Detalhes do App” preencha os dados do seu App, conforme a figura “Detalhes do App”. Role a página para baixo e marque o aceite dos termos de uso, conforme a figura “Aceite dos termos de uso” e clique no botão “Criar app” no canto inferior direito da página. Aguarde a aplicação ser criada. Dependendo da sua conexão de Internet esse processo pode demorar um pouquinho.

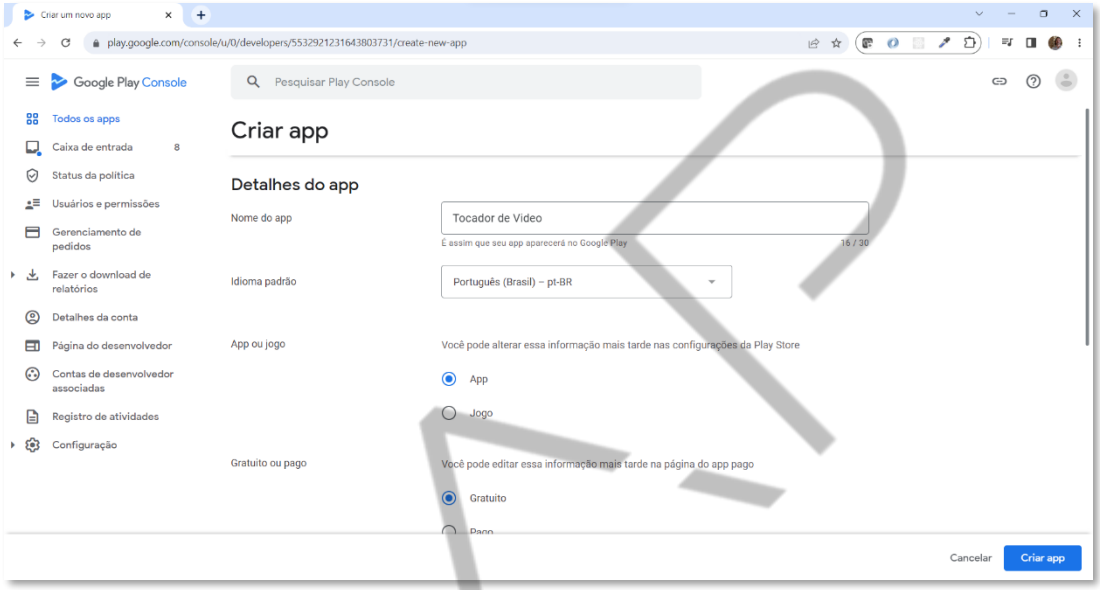

Figura 19 – Detalhes do App
Fonte: Elaborado pelo autor (2023)

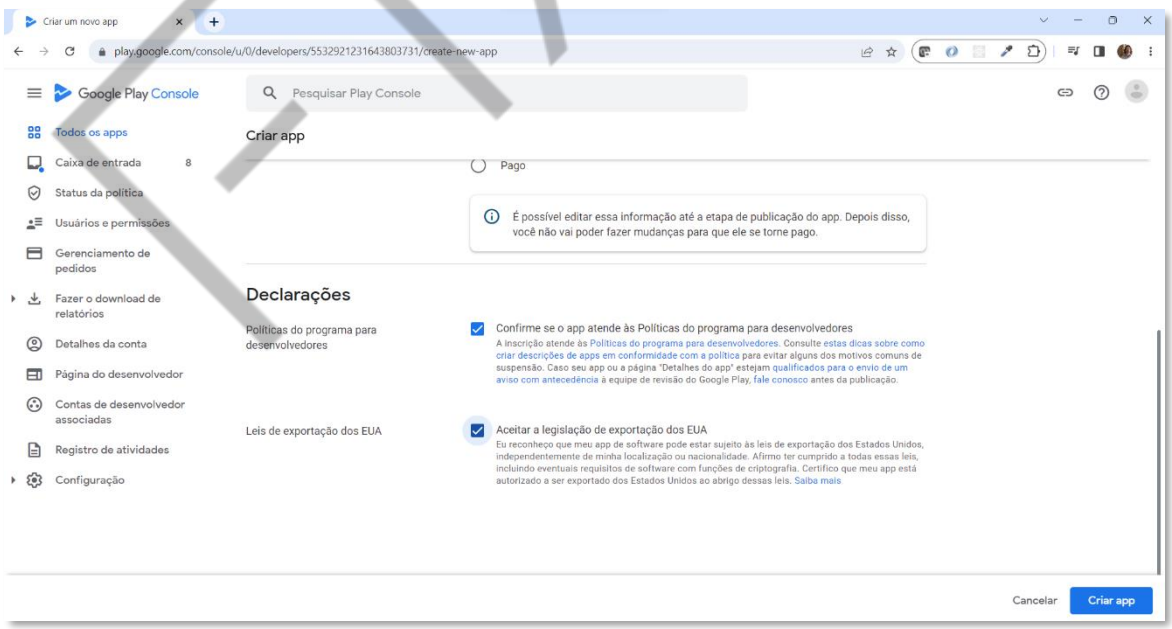

Figura 20 – Aceite dos termos de uso
Fonte: Elaborado pelo autor (2023)

Assim que o App for criado, nós seremos direcionados para a página “Painel”, onde deveremos seguir um passo a passo muito bem explicado sobre como fazer a publicação, então, role a página para baixo até onde se vê o título “Configurar o App”, conforme a figura “Passos para publicação”.

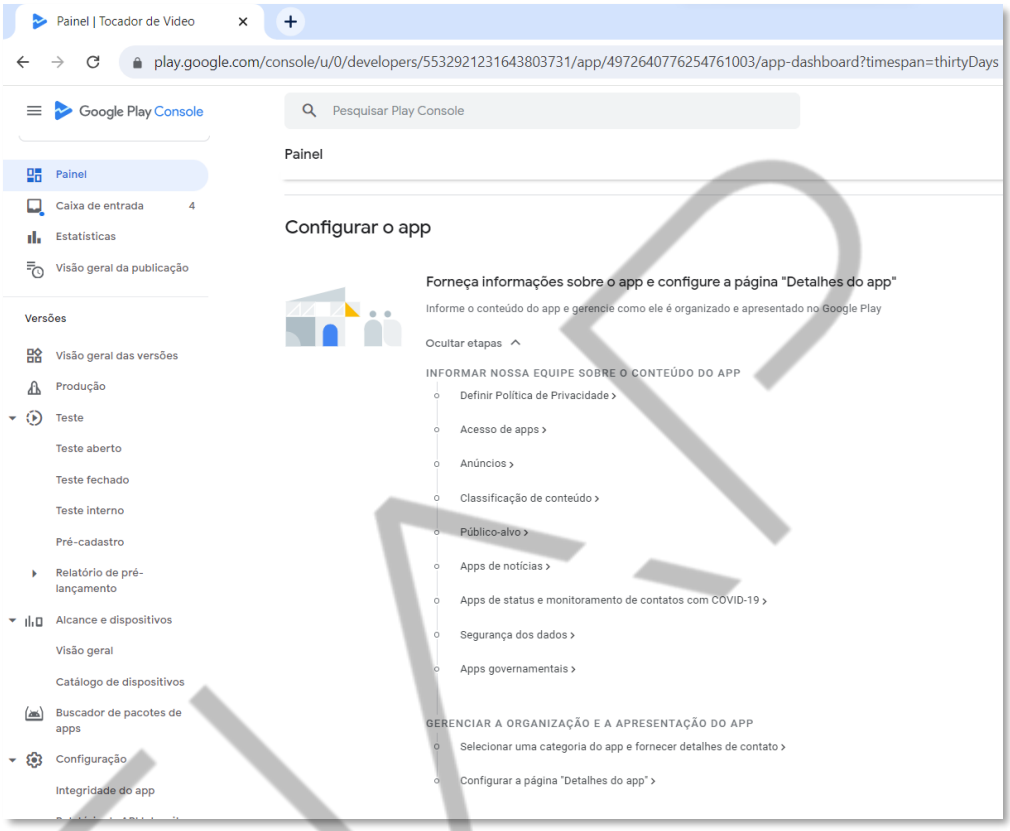

Figura 21 – Passos para publicação
Fonte: Elaborado pelo autor (2023)

Acesse cada um dos passos da lista e forneça os dados solicitados. Você será convidado a responder um questionário sobre qual a finalidade do aplicativo, público-alvo, se possui anúncios etc. Ao finalizar o passo a passo a sua aplicação será avaliada pela equipe do Google e se tudo estiver OK, ela será disponibilizada para os usuários do mundo todo.

REFERÊNCIAS

DEVELOPERS. **Animação**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/animation?hl=pt-br>. Acesso em: 10 jul. 2023.

DEVELOPERS. **MediaPlayer**. 2023. Disponível em: <https://developer.android.com/reference/kotlin/android/media/MediaPlayer>. Acesso em: 10 jul. 2023.

DEVELOPERS. **ExoPlayer**. 2023. Disponível em: <https://developer.android.com/reference/androidx/media3/exoplayer/ExoPlayer>. Acesso em: 10 jul. 2023.

DEVELOPERS. **Efeitos colaterais no compose**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/side-effects?hl=pt-br>. Acesso em: 10 jul. 2023.

DEVELOPERS. **Sobre os Android App Bundles**. 2023. Disponível em: <https://developer.android.com/guide/app-bundle?hl=pt-br>. Acesso em: 10 jul. 2023.