LAB 11 – Video Playback on Android

At the end of this lab, students should be able to:

- Demonstrate the usage of VideoView and MediaCotnroller classes through the creation of a video playback application.

Android VideoView Class

The VideoView class has a wide range of methods that may be called in order to manage the playback of video. Some of the more commonly used methods are as follows:
•**setVideoPath(String path)** – Specifies the path (as a string) of the video media to be played. This can be either the URL of a remote video file or a video file local to the device.
•**setVideoUri(Uri uri)** – Performs the same task as the setVideoPath() method but takes a Uri object as an argument instead of a string.
•**start()** – Starts video playback.
•**stopPlayback()** – Stops the video playback.
•**pause()** – Pauses video playback.
•**isPlaying()** – Returns a Boolean value indicating whether a video is currently playing.
•**setOnPreparedListener(MediaPlayer.OnPreparedListener)** – Allows a callback method to be called when the video is ready to play.
•**setOnErrorListener(MediaPlayer.OnErrorListener)** - Allows a callback method to be called when an error occurs during the video playback.
•**setOnCompletionListener(MediaPlayer.OnCompletionListener)** - Allows a callback method to be called when the end of the video is reached.
•**getDuration()** – Returns the duration of the video. Will typically return -1 unless called from within the OnPreparedListener() callback method.
•**getCurrentPosition()** – Returns an integer value indicating the current position of playback.
•**setMediaController(MediaController)** – Designates a MediaController instance allowing playback controls to be displayed to the user.

Android MediaController Class

The MediaController will then provide a set of controls allowing the user to manage the playback (such as pausing and seeking backwards/forwards in the video time-line).

Some of the key methods of this class are as follows:
•**setAnchorView(View view)** – Designates the view to which the controller is to be anchored. This designates the location of the controls on the screen.
•**show()** – Displays the controls.
•**show(int timeout)** – Controls are displayed for the designated duration (in milliseconds).
•**hide()** – Hides the controller from the user.
•**isShowing()** – Returns a Boolean value indicating whether the controls are currently visible to the user.

VideoPlayback Project- An Example

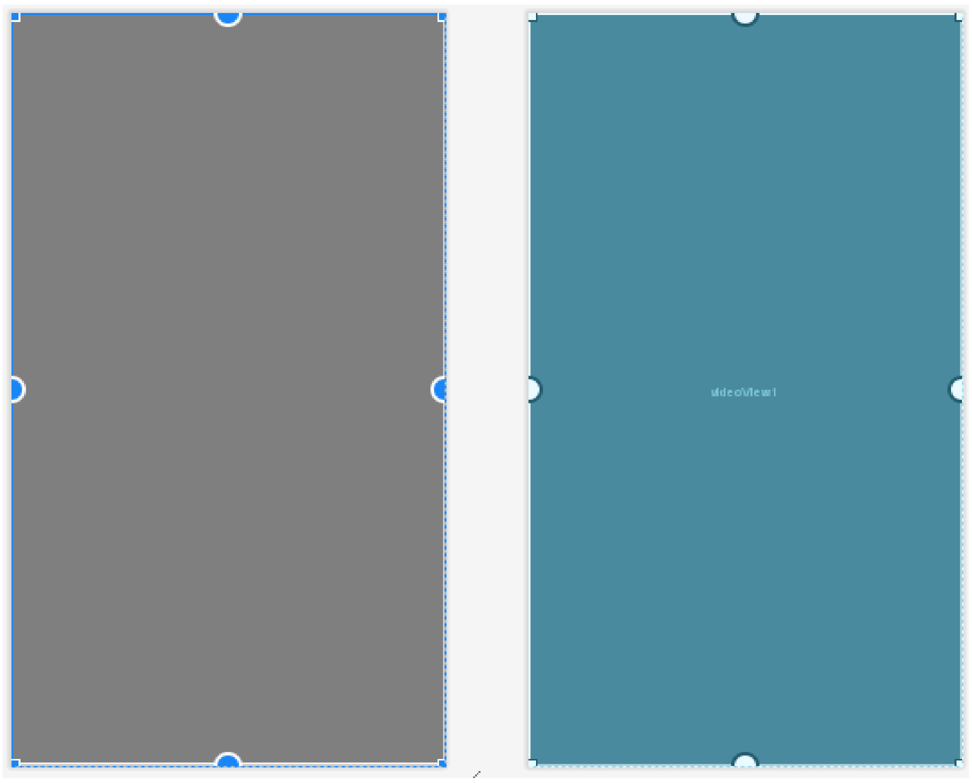
a) Creating the Example Application

i. Go to File -> New Project
ii. Choose **Empty Activity** template, then click Next
iii. Enter *VideoPlayer* into the Name field and specify com.ebookfrenzy.videoplayer as the package name.
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

b) Designing the VideoPlayer Layout

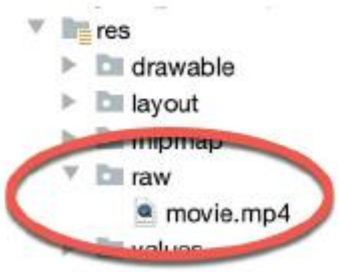
The user interface for the main activity will consist solely of an instance of the VideoView class. Use the Project tool window to locate the app -> res -> layout -> activity_main.xml file, double-click on it, switch the Layout Editor tool to Design mode and delete the default TextView widget.

i. From the Widgets category of the Palette panel, drag and drop a VideoView instance onto the layout so that it fills the available canvas area as shown in figure below.
ii. Using the Attributes panel, change the layout_width and layout_height attributes to *match_constraint* and *wrap_content* respectively. Also, remove the constraint connecting the bottom of the VideoView to the bottom of the parent ConstraintLayout. Finally, change the ID of the component to *videoView1*.



c) Importing the Video File

i. Within Android Studio, locate the *res* folder in the Project tool window, right-click on it and select the *New -> Directory* menu option and enter *raw* into the name field before clicking on the OK button. Using the file system navigator for your operating system, locate the movie.mp4 file that you have downloaded and copy it. Returning to Android Studio, right-click on the newly created raw directory and select the Paste option to copy the video file into the project. Once added, the raw folder should match within the Project tool window:



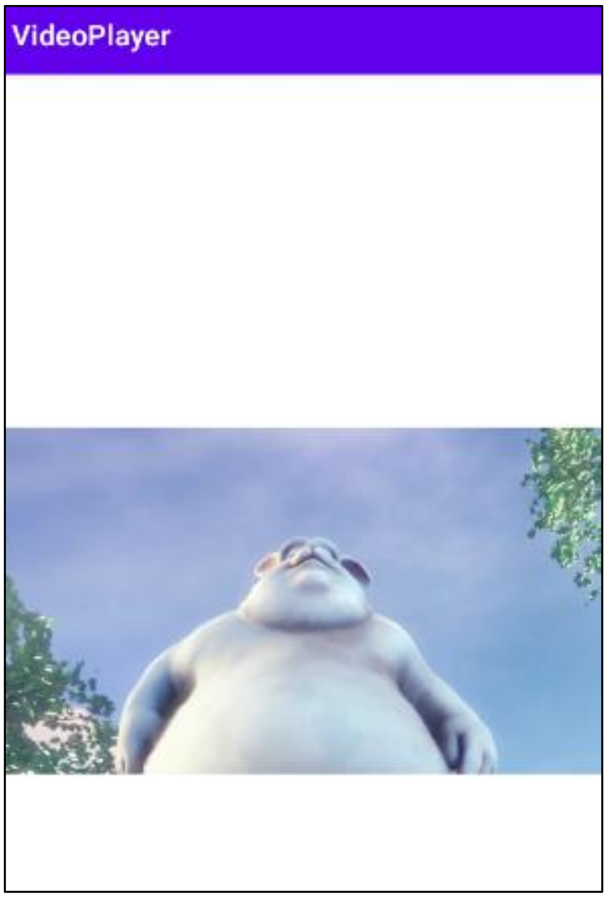
d) Configuring the Video View

i. The next step is to configure the VideoView with the path of the video to be played and then start the playback. This will be performed when the main activity has initialized, so load the *MainActivity.java* file into the editor and modify it as outlined in the following:

```
package com.ebookfrenzy.videoplayer;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
import android.net.Uri;
public class MainActivity extends AppCompatActivity {
private VideoView videoView;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_video_player);
configureVideoView();
}
private void configureVideoView() {
videoView =
findViewById(R.id.videoView1);
videoView.setVideoURI(Uri.parse("android.resource://"
+ getPackageName() + "/" + R.raw.movie));
videoView.start();
}
}
```

All that this code does is obtain a reference to the VideoView instance in the layout, assigns to it a URI object referencing the movie file located in the raw resource directory and then starts the video playing.

Test the application by running it on a physical Android device. After the application launches there may be a short delay while video content is buffered before the playback begins as below.



This provides an indication of how easy it can be to integrate video playback into an Android application. Everything so far in this example has been achieved using a VideoView instance and three lines of code.

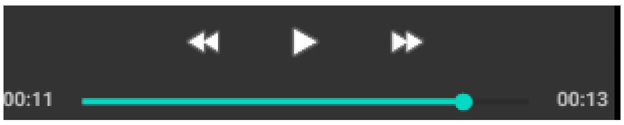
e) <u>Adding the Media Controller to the Video View</u>

As the VideoPlayer application currently stands, there is no way for the user to control playback. As previously outlined, this can be achieved using the MediaController class.
To add a controller to the VideoView, modify the configureVideoView() method once again:

```
package com.ebookfrenzy.videoplayer;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
import android.net.Uri;
import android.widget.MediaController;
public class MainActivity extends AppCompatActivity {
```

```
private VideoView videoView;
private MediaController mediaController;
..
private void configureVideoView() {
final VideoView videoView =
findViewById(R.id.videoView1);
videoView.setVideoURI(Uri.parse("android.resource://"
+ getPackageName() + "/" + R.raw.movie));
mediaController = new MediaController(this);
mediaController.setAnchorView(videoView);
videoView.setMediaController(mediaController);
videoView.start();
}
}
```

When the application is launched with these changes implemented, tapping the VideoView canvas will cause the media controls to appear over the video playback.
These controls should include a seekbar together with fast forward, rewind and play/pause buttons.
After the controls recede from view, they can be restored at any time by tapping on the VideoView canvas once again. With just three more lines of code, our video player application now has media controls as shown in figure below:



f) Setting up the onPreparedListener

As a final example of working with video based media, the activity will now be extended further to demonstrate the mechanism for configuring a listener.
In this case, a listener will be implemented that is intended to output the duration of the video as a message in the Android Studio Logcat panel. The listener will also configure video playback to loop continuously:

```
package com.ebookfrenzy.videoplayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
import android.net.Uri;
import android.widget.MediaController;
import android.util.Log;
import android.media.MediaPlayer;
public class MainActivity extends AppCompatActivity {
private VideoView videoView;
private MediaController mediaController;
String TAG = "VideoPlayer";
private void configureVideoView() {
final VideoView videoView =
findViewById(R.id.videoView1);
videoView.setVideoURI(Uri.parse("android.resource://"
+ getPackageName() + "/" + R.raw.movie));
MediaController mediaController = new
MediaController(this);
mediaController.setAnchorView(videoView);
videoView.setMediaController(mediaController);
videoView.setOnPreparedListener(new
MediaPlayer.OnPreparedListener() {
@Override
public void onPrepared(MediaPlayer mp) {
mp.setLooping(true);
Log.i(TAG, "Duration = " +
videoView.getDuration());
}
});
videoView.start();

}
}
```

Now just before the video playback begins, a message will appear in the Android Studio Logcat panel that reads along the lines of the following and the video will restart afterplayback ends:

```
11-05 10:27:52.256 12542-12542/com.ebookfrenzy.videoplayer
I/VideoPlayer: Duration = 13504
```