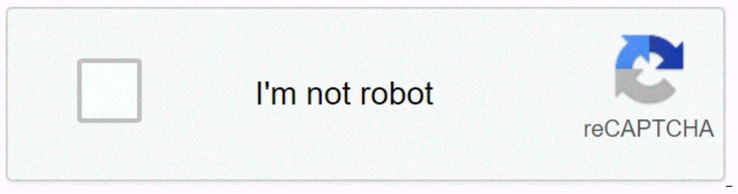

Continue

Android 360 video player library

Flutter plug-in for streaming 360° videos to iOS and Android This is a Flutter plug-in to play 360° videos via a remote URL. The iOS player uses the open source Google VR SDK for iOS Android Player uses open source Google VR SDK for Android Add video_player_360: ^0.1.4 in pubspec.yaml. import library: import package:video_player_360/video_player_360.dart; Video playback: Waiting videoPlayer360.playVideoURL(ENTER_360_VIDEO_URL_HERE); This plug-in won't work if the flutter project was created with a Swift flag. 360 Video Player for Android using the 3D Rajawali library. The Player can play 360 videos stored in the SD card as well as via a URL. ExoPlayer is a widespread library for running personalized video players in Android apps, as it is an open source library supported by Google. This is an alternative to the Android MediaPlayer API providing many more out-the-box features as well as supporting many video formats and resources. Therefore, we can develop an advanced and customized video player that avoids re-bringing the wheel out. One of the many advanced capabilities of ExoPlayer allows you to integrate 360 video playback. This, combined with the Android motion sensor hardware, improves the VR experience in multimedia applications without the need for special handsets or peripherals; only with a smartphone or tablet. There are many resources and information for learning and research on ExoPlayer, such as a developer guide and an official blog. However, none of them is enough to understand how to properly make a setting and how to code at least a basic VR player. The code lab does not cover this feature and is a little outdated because the exoPlayer API has since changed dramatically, so it uses outdated AIIs. Also, the demo application performs this and many other features in single-lite architecture, so the code is not so clean and may be confusing, so you can get stuck with a few questions. So here's a simple guideline for minimalist VR application implementation as shown in the following video (Try not to scare those monkeys 😆 🙈)First, How is stated ExoPlayer from some version u second is th by 10th-century API, so we are on the same page add traces of envy u build.gradle datoteci:Additionally, it is necessary to target Javu 8, because the library benefits from implementation of the implemented interfacea methods, or i to escape some vague problems. In the android section of the build.gradle app file, we need to add the following: To enable our app to download media from remote sources, remember to request internet permissions for the app in the AndroidManifest file. Now we can declare the PlayerView component in the layout as if it were for standard video, but this time we define spherical_gl_surface_view as the type of surface to load video format 360, allowing us to explore video in all directions instead 2D video. If we do not use this setting, then an outcast video with an unexpected juctapostion, a kind of surreal image that could cause users to experience dizziness at the beginning of this article. Then we need to use this view in our Activity or Fragment and only 2 components: SimplePlayer and MediaSource. The library offers many factories to support many existing types of media and resources that could store our app with multimedia data such as HLS, DASH, Ogg, WebM, FLV and so on. For this sample, we will use a common downloadable MP4 file so that you can configure ProgressiveMediaSource with defaultDataSource, which supports multiple Schema uri for local resources in addition to remote resources via HTTP. Create them as follows:Something else is that there are 4 types of stereoscocular modes for 360/3D/VR videos: Monoscopic, Top-Bottom, Left-Right and one where the left and right eyes have separate eyes. If your video contains Spherical Video V2 metadata, stereo mode can be detected automatically, otherwise it uses the default. Therefore, it is important to set the appropriate default mode for the media that we want to play (for our sample video is below) according to the SphericalGLSurfaceView contract with the setDefaultStereoMode(C.STEREO_MODE_TOP_BOTTOM). If the default mode is incorrect, you could get a video with strange shapes, such as a kaleidoscope. Finally, we can initialize playback by entering SimpleExoPlayer, preparing it with a handy MediaSource, and tying it to the PlayerView component. It is also very important to call playerView.onResume() to enable Android motion sensors in SphericalGLSurfaceView and use built-in hardware as a gyroscope to navigate the scene when we move a mobile device. If the sdk version of the software currently running on the device is less than 24 (the so-called API level that is not supported by multiple windows), the procedure should be called using the hook lifecycle onResume method. Otherwise, it needs to be called using the Startup app because the Player app can be visible — but not active — in split window mode. Here is the help method that performs this routine:In addition, as soon as the video is no longer viewed, we need to free up the player to free up hardware resources and undo its reference. This can happen when the lifecycle status of NaPaused or Stop is reached, depending on the Android API level. Composition of all the pieces in the pattern Activity we get the following codeConclusive and follow-up workAnd that's all people! Thus, with only 70 lines of code (Kotlin) is enough to develop a simple VR application. Of course there are many specific capabilities, configurations, UI appearance & feel and limitations by ExoPlayer allows implementation to meet the requirements of your application. On the final note, I recommend that you bathe in the source code and keep a close eye on javadoc. For a thorough understanding and mastery of ExoPlayer, here you have some resources If you liked this article, clap and please, please. Hi! Stay in 😀Web site | Twitter I've been impressed by 360 videos on YouTube since recently. I would like to develop a sample Android application that can play 360 video and can pan/swipe with it along with easy use of accelerator/gyroscope. Some questions: Which file format is 360 video? Where can I download a sample of 360 video? Is it even possible to use an Android library to play 360 video? If so, which player should use to play 360 video? How can I download pan/swipe for 360 homeman's video player? Is it possible to play an existing YouTube 360 video using Android native Player? And at the same time you can handle pan/swype/gyro? Provide a code pattern. Thanks! Augmented and virtual reality, though still relatively new, quickly became popular for apps, including gaming and education. Earlier I showed you how to install Cardboard using Android SDK and how to create a panoramic image viewer. This post will show you how to use 360-degree video in apps. SetupCoup you start building an app to view videos, you will need to clone the Android cardboard SDK on your computer via Git. For instructions on this, see the previous article in this series. For our sample, create a new Android project with a minimum SDK api 19 (KitKat) and use the Blank Activity template. When you create a basic project, you will need to copy commonwidget and videowidget folders from the SDK card to the root of your project. Once these directories are moved, you'll need to include them as modules in your project by editing the settings.gradle file to make them look like the next snip. vključujejo ':app', :common, commonwidget, videowidgetKončno, Vključite te i druge potrebne knjižnice u vaš projekt tako da dodate u datoteku build.gradle modula aplikacije u datoteku build.gradle pod dependencies node.dependencies { compile 'com.android.support:appcompat-v7:25.0.0' compile project(':common') compile project(':commonwidget') compile project(:commonwidget) compile project(:commonwidget) :videowidget') compile 'com.google.android.exoplayer:exoplayer:r1.5.10' compile 'com.google.protobuf.nano:protobuf-javanano:3.0.0-alpha-7' }Opazit čete da smo dodali knige Protocol Buffers iz Googlea, to vam pomaže u upravljanju runtime virsama na napravu, i ExoPlayer, a to je knjižnica video playera, nastala googleom, na kateri je nastala komponenta VrVideoView. Both libraries require an SDK card and you may have noticed that the ExoPlayer version used from the first release is not the second, so it can cause conflicts if you use ExoPlayer v2 in your own projects. Then, we'll activity_main.xml update the file for the sample project to include VrVideoView, the search bar, and the activity_main.xml we'll work with later. <?xml version=1.0 encoding=utf-8? > <LinearLayout xmlns:android= android:layout_width=match_parent android:layout_height=match_parent <com.google.vr.sdk.widgets.video.VrVideoView android:id=@+id/video_view android:layout_width=match_parent android:layout_height=250dp></com.google.vr.sdk.widgets.video.VrVideoView><SeekBar:android=@+id/seek_bar android:layout_height=32dp android:layout_width=match_parent style=?android:attr/progressBarStyleHorizontal></SeekBar> <Button android:id=@+id/btn_volume android:layout_width=wrap_content android:layout_height=wrap_content android:text=Volume To><Button/ggle; When you are finished setting up cardboard libraries and you have created a layout, which we will use, it's time to jump into java code. Working with cardboard and VR VideoNakar we can start writing all the code that controls the status of playback, position and upload to the video file 360, we will have to determine the source of our video. For this tutorial we will simply create a folder of resources under the main directory, and place 360 video there. While there are some sources for 360 videos online, I included a short video of the public domain from Sea World returning a sea turtle to the ocean in an accompanying GitHub project for this tutorial. Initialize and StructureNow to have a video file to play, open the MainActivity class. First, you will need to log on and initialize the View items specified by the layout file, as well as two boolean values to track the status of muted and playback/pause. In addition, we will install OnClickListener on our Volume Button object.public class MainActivity extends AppCompatActivity { private VrVideoView mVrVideoView; private SeekBar mSeekBar; private Button mVolumeButton; private boolean mIsPaused; private boolean mIsMuted; @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); initViews(); } private void initViews() { mVrVideoView = (VrVideoView) findViewById(R.id.video_view); mSeekBar = (SeekBar) findViewById(R.id.seek_bar); mVolBuumetton = (() Button) findViewById(R.id.btn_volume); mVolumeButton.setOnClickListener(new View.OnClickListener() { @Override public void onClick(View view) { onVolumeTo ggleClicked(); }); } public void playPause() { } public void onVolumeToggleClicked() { } }Next, create a new inner class that extends VrVideoEventListener. This class will have five methods that we can implement for our simple video viewer.private class ActivityEventListener expanded VrVideoEventListener { @Override public void onLoadSuccess() { super.onLoadSuccess(); } @Override public void onLoadError(String errorMessage) { super.. onLoadError(errorMessage); } @Override public void onClick() { super.onClick(); } @Override public void onNewFrame() { super.onNewFrame(); } @Override public void onCompletion() { super.onCompletion(); } You will also need to implement SeekBar.OnSeekBarChangeListener in your class and create methods for this interface.public class MainActivity extends AppCompatActivity implements { ... public emptiness emptiness { } public void naVolumeToggleClicked() { } @Override public void onProgressChanged(SeekBar seekBar, int i, boolean b) { } }}} When you purchased here the new inner class i SeekBar implementationu, connect them to vrVideoView i SeekBar, at the end of the initViews() method defined above.mVrVideoView.setEventListener(new ActivityEventListener()); mSeekBar.setOnSeekBarChangeListener(to); Another piece of installation needs to be taken care of. You'll need to address the Android activity lifecycle by supporting onPause(, onResume() and onDestroy() modes to pause or resume staging in VrVideoView or to stop it completely. You will also need to monitor the pause status methods.@Override these protected voids onPause() { super.onPause(); mVrVideoView.pauseRendering(); mIsPaused = true; } @Override the filloid onResume() { super.onResume(); mVrVideoView.resumeRendering(); mIsPaused = false; } @Override protected void onDestroy() { mVrVideoView.shutdown(); super.onDestroy(); } Now that the initial setup of our exercise class is complete, you can switch to a more interesting topic: uploading a video, monitoring playback, and customizing the user experience. Starting and controlling VrVideoView Since uploading a 360 video can take anywhere from a split second to a few seconds, you'll want to handle uploading the video through a background task. Let's start by creating a new AsyncTask that will create a new VrVideoView.Options object, set the input type to match the formatting of our video (in the case of this tutorial, TYPE_MONO), and then upload our video from the resource directory.class VideoLoaderTask expands AsyncTask<Void, void,= boolean=> {@Override protected BooleanInroundroundround(Void... voids) { try { VrVideoView.Options options = new VrVideoView.Options(); options.inputType = VrVideoView.Options.TYPE_MONO; mVrVideoView.loadVideoFromAsset(seaturtle.mp4, options); } catch(IOException e) { //Handle exception } true return; } } Then go to the onCreate method() and create a new instance of this task, and then call the execut(s) to run. While there is some more that should be done to properly maintain this task, we will use it locally in this method of simplicity and do not worry about AsyncTask lifecycle considerations. VideoLoaderTask mBackgroundVideoLoaderTask = new VideoLoaderTask(); mBackgroundVideoLoaderTask.execute(); At this point, you should be able to run your app and watch video playback 360 in Cardboard video view. Now that this is working, let's add another utility for our user. Return to the ActivityEventListener object that you created earlier in this tutorial, as we will want to make meat from some methods. When a video loads successfully, we need to set the maximum value for our SeekBar, as well as monitor the playback/pause status of our video.@Override public void atLoadSuccess() { super.onLoadSuccess(); mSeekBar.setMax((int) </Void,>mIsPaused = false; }When the video is played, we will update this SeekBar through onNewFrame() and reset the video to its starting position in onCompletion(). Finally, in onClick(), we will trigger our game/switch switching method.@Override public void at Click() { playPause(); } @Override public emptiness onNewFrame() { super.onNe mSeekBar.setProgress((int) mVrVideoView.getCurrentPosition()); } @Override public emptiness onCompletion() { //Restart the video, He is allowed to take a look at mVrVideoView.seekTo(0); } While updating our SeekBar-based playback is important, we'll also want to allow the user to change where it is located in the video by interacting with SeekBar. We can do this by using seekbar.OnSeekBarChangeListen earlier.@Override er interface, which we implemented into the public void at ProgressChanged(SeekBar seekBar, int progress, boolean fromUser) { if(fromUser) { mVrVideoView.seekTo(progress); } To round up our vrVideoView controls, we will need to perform play/pause and switch the volume method.public void playPause() { if(mIsPaused) { mVrVideo.playVideo(); } else { mVrVideo.pauseVideo(); } mIsPaused = !mIsPaused; } public void naVolumeToggleClicked() { mIsMuted = !mIsMuted; mVrVideoView.setVolume(mIsMuted ? 0.0f : 1.0f); } At this time, you should have a fully functioning and interactive 360 video player in the app. We can fix this by working with Android naSaveInstanceState() and onRestoreInstanceState() to save and reset the status of our VrVideoView.private static final String STATE_PROGRESS = state_progress; private static end string STATE_DURATION = state_duration; @Override naSaveInstanceState(Bundle outState) { outState.putLong(STATE_PROGRESS, mVrVideoView.getCurrentPosition()); outState.putLong(STATE_DURATION, mVrVideoView.getDuration()); super.onSaveInstanceState(outState); } @Override protected void onRestoreInstanceState(savedInstanceState) { super.onRestoreInstanceState(savedInstanceState); long progress = savedInstanceState.getLong(STATE_PROGRESS); mVrVideoView.seekTo(progress); mSeekBar.setMax((int) savedInstanceState.getLong(STATE_DURATION)); mSeekBar.setProgress((int) progress); } When the device is rotated, have your video returned to its original position and your user can continue with their continuous experience. Conclusion While there are some small details that need to be considered for the use of VrVideoView SDK cardboard, heavy parts such as actual playback and optimization, for you deal with a component that is easy to implement. They should now be able to add 360 videos to their media apps to provide users with an interesting feature that enriches their experience. In the next tutorial of this series, we will focus on Google's new VR experience called Daydream and How to Use a Paired Controller with Meanwhile, take a look at some of our other on Android virtual reality and augmented reality! Reality!