Media Player

Phase II: Report

Group 21
1155175490 LI Hoi Hin
1155175164 LU Man Ho
1155216903 MATTHAEI Casper Ludwig

Department of Computer Science and Engineering
The Chinese University of Hong Kong

May 6, 2024

1. Introduction

Our media player application provides a comprehensive solution for playing various media files, including videos and audio, with additional features like picture-in-picture mode, subtitle support, sharing functionality, and reading files from storage.

We have taken some real-world android application references. For example, VLC Media Player and MX Player. The former is a free and open source cross-platform multimedia player that plays most multimedia files as well as discs, devices, and network streaming protocols, while the latter is a Powerful video and music player with advanced hardware acceleration and subtitle support.

2. Layout flow

The home activity of the application holds the two main navigation fragments: audio and video fragments. When starting the application for the first time a prompt asking for permission to storage and media is shown. The first fragment that is shown is the audio fragment showing a list of media items collected from the phone's storage. From here the user can navigate between the two main fragments, the video fragment and the audio fragment shown on the left hand side of figure 1. If the user presses one of the media items an intent is sent to either the respective mediaPlayer activity. As seen in figure 1 the videoPlayer and audioPlayer includes a timeline scrubber, play and pause and for audio playback next and previous audio file, all pages work in both portrait and landscape mode.

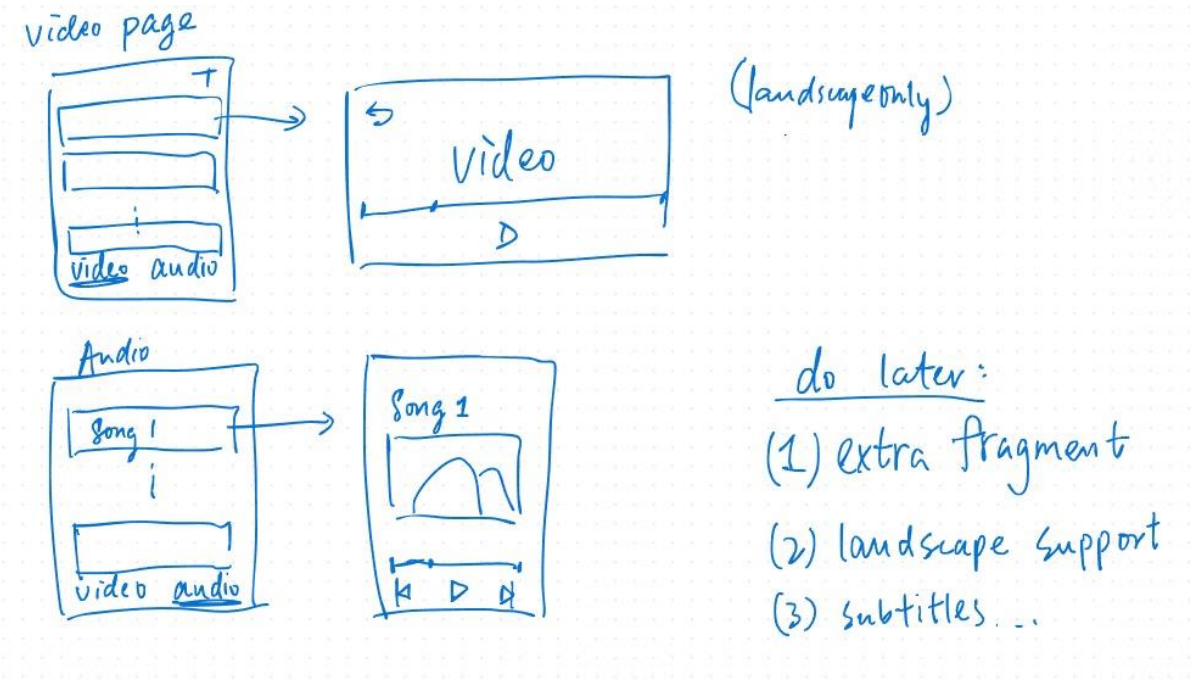

figure 1: Storyboard

In figure 2 the navigation is visualized. The audio fragment is tied to the main activity, pressing backspace results in closing the application. When moving to the videofragment the audio fragment is added to the backstack, going back from the video fragment directs the user to the audio fragment. When moving to the audioPlayer and videoPlayer activity the main activity is pushed to the backstack with its respective fragment. Going back from videoPlayer directs the user to the video fragment and going back from audioPlayer directs the user to the audio fragment.

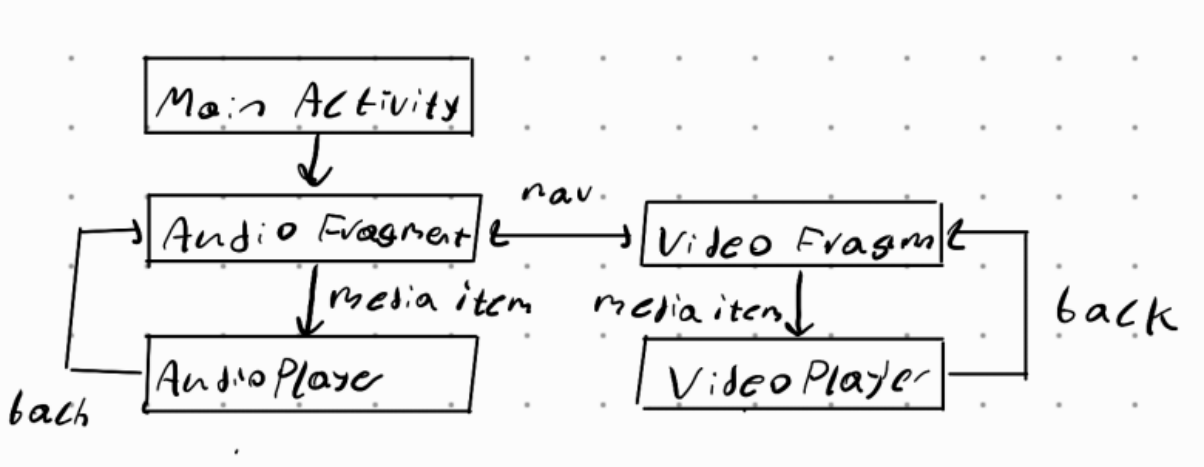

figure 2: Navigation

3. User guide

The application was created using Java and developed using Windows and Mac. The hedgehog 2023.1.1 version of Android Studio was used and the main testing platform was the Google Pixel 3 emulator in Android Studio. The API level of the emulated android phone used is API level 34.

Before starting the application some media files have to be added to the phone's storage. These media files have to be in Audio or Video file formats. These media files should be added to either the downloads folder in the phones internal or external storage or the respective audio and video folders. If no audio and video files are in the phone's storage the audio then the audio and video fragments will be empty. It is also sufficient to download audio files and video files from the internet and then start the app without moving the files to any specific folders.

4. Features

4.1. File management

The two start fragments, Audio fragment and Video fragment are the main methods of file management. These fragments use a recycler list to display the media files to the user. Each media file displays an image, the title of the file and the duration of the media content. The files are collected from the phone's internal storage using the MediaStore library. The file path, title, duration are collected using the MediaStore library and are saved to a class object in the application. When the user clicks on an audio file in the recycler list, an intent starts the audioPlayback activity and the position of the media file as well as the list of all existing media class objects is sent to the activity. If a video file is pressed by the user an intent starts the videPlayback activity and the video URI is sent to the activity.

4.2. Audio Playback

The audio functions retrieve audio files from the device's media store, represent these files as MediaModel instances, display these instances in a RecyclerView, and handle user interactions with this RecyclerView. It supports a wide range of audio formats, e.g. MP3, OGG, WAV etc. It also supports basic functions like play and pause the song, play previous/next song, progress bar, background music playback and share function.

4.3. Video Playback

The video player in our application supports both portrait and landscape mode, along with picture-in-picture mode which is activated when the user presses the “Home” button on his Android phone. The progress of the video is conserved when switching between modes using shared preference. The volume slider is only visible when the player clicks on the player and the detail control interface shows up in portrait and landscape mode. Subtitles are displayed if the video files have included subtitles internally.

4.4 Share functionality

In both the videoPlayer and audioPlayer a share functionality is included. If pressed the file URI is shared to an external application. This is done using the ShareSheet library, the file URI is packed up into an explicit intent using the ACTION_SEND flag. The intent then calls createChooser to display the ShareSheet for selecting the destination of the data between available applications. For the Audioplayer we set the Intent data type to audio and video is used for the Videoplayer.

5. Reflections

There are several changes made compared to the initial proposal when implementing the features of the video player. Initially we did not fully grasp the capacity and power of the Exoplayer library, and we had the wrong impression that every UI element of the player must be implemented manually by the developers. However, we noticed that the Exoplayer already manages the player user interface (PlayerControllerView), and it already includes a timeline scrubber and basic video player structure. Therefore, we did not include the library “PreviewSeekBar” mentioned in the proposal section 3.1.2. to implement the seek bar for video progress navigation.

Furthermore, since the target Android version is above 12, we have only implement the picture-to-picture mode by setting the “setAutoEnterEnabled” flag in PictureInPictureParams.Builder() to true, and specifying the app that it supports picture-in-picture mode in AndroidManifest.xml.

Another change compared to the initial proposal is the file system management. In the proposal it was planned that the management would include application file management and that the user had to import the media from the phone's storage into specific folders in the application. For the current file management the application instead reads all available music and video files from the phone storage, eliminating the need to import the files manually.