// Copyright 2013 The Flutter Authors. All rights reserved.

// Use of this source code is governed by a BSD-style license that can be

// found in the LICENSE file.

// ignore\_for\_file: public\_member\_api\_docs

import 'dart:async';

import 'dart:io';

import 'package:flutter/foundation.dart';

import 'package:flutter/material.dart';

import 'package:image\_picker/image\_picker.dart';

import 'package:mime/mime.dart';

import 'package:video\_player/video\_player.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return const MaterialApp(

title: 'Image Picker Demo',

home: MyHomePage(title: 'Image Picker Example'),

);

}

}

class MyHomePage extends StatefulWidget {

const MyHomePage({super.key, this.title});

final String? title;

@override

State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

List<XFile>? \_mediaFileList;

void \_setImageFileListFromFile(XFile? value) {

\_mediaFileList = value == null ? null : <XFile>[value];

}

dynamic \_pickImageError;

bool isVideo = false;

VideoPlayerController? \_controller;

VideoPlayerController? \_toBeDisposed;

String? \_retrieveDataError;

final ImagePicker \_picker = ImagePicker();

final TextEditingController maxWidthController = TextEditingController();

final TextEditingController maxHeightController = TextEditingController();

final TextEditingController qualityController = TextEditingController();

Future<void> \_playVideo(XFile? file) async {

if (file != null && mounted) {

await \_disposeVideoController();

late VideoPlayerController controller;

if (kIsWeb) {

controller = VideoPlayerController.networkUrl(Uri.parse(file.path));

} else {

controller = VideoPlayerController.file(File(file.path));

}

\_controller = controller;

// In web, most browsers won't honor a programmatic call to .play

// if the video has a sound track (and is not muted).

// Mute the video so it auto-plays in web!

// This is not needed if the call to .play is the result of user

// interaction (clicking on a "play" button, for example).

const double volume = kIsWeb ? 0.0 : 1.0;

await controller.setVolume(volume);

await controller.initialize();

await controller.setLooping(true);

await controller.play();

setState(() {});

}

}

Future<void> \_onImageButtonPressed(

ImageSource source, {

required BuildContext context,

bool isMultiImage = false,

bool isMedia = false,

}) async {

if (\_controller != null) {

await \_controller!.setVolume(0.0);

}

if (context.mounted) {

if (isVideo) {

final XFile? file = await \_picker.pickVideo(

source: source, maxDuration: const Duration(seconds: 10));

await \_playVideo(file);

} else if (isMultiImage) {

await \_displayPickImageDialog(context,

(double? maxWidth, double? maxHeight, int? quality) async {

try {

final List<XFile> pickedFileList = isMedia

? await \_picker.pickMultipleMedia(

maxWidth: maxWidth,

maxHeight: maxHeight,

imageQuality: quality,

)

: await \_picker.pickMultiImage(

maxWidth: maxWidth,

maxHeight: maxHeight,

imageQuality: quality,

);

setState(() {

\_mediaFileList = pickedFileList;

});

} catch (e) {

setState(() {

\_pickImageError = e;

});

}

});

} else if (isMedia) {

await \_displayPickImageDialog(context,

(double? maxWidth, double? maxHeight, int? quality) async {

try {

final List<XFile> pickedFileList = <XFile>[];

final XFile? media = await \_picker.pickMedia(

maxWidth: maxWidth,

maxHeight: maxHeight,

imageQuality: quality,

);

if (media != null) {

pickedFileList.add(media);

setState(() {

\_mediaFileList = pickedFileList;

});

}

} catch (e) {

setState(() {

\_pickImageError = e;

});

}

});

} else {

await \_displayPickImageDialog(context,

(double? maxWidth, double? maxHeight, int? quality) async {

try {

final XFile? pickedFile = await \_picker.pickImage(

source: source,

maxWidth: maxWidth,

maxHeight: maxHeight,

imageQuality: quality,

);

setState(() {

\_setImageFileListFromFile(pickedFile);

});

} catch (e) {

setState(() {

\_pickImageError = e;

});

}

});

}

}

}

@override

void deactivate() {

if (\_controller != null) {

\_controller!.setVolume(0.0);

\_controller!.pause();

}

super.deactivate();

}

@override

void dispose() {

\_disposeVideoController();

maxWidthController.dispose();

maxHeightController.dispose();

qualityController.dispose();

super.dispose();

}

Future<void> \_disposeVideoController() async {

if (\_toBeDisposed != null) {

await \_toBeDisposed!.dispose();

}

\_toBeDisposed = \_controller;

\_controller = null;

}

Widget \_previewVideo() {

final Text? retrieveError = \_getRetrieveErrorWidget();

if (retrieveError != null) {

return retrieveError;

}

if (\_controller == null) {

return const Text(

'You have not yet picked a video',

textAlign: TextAlign.center,

);

}

return Padding(

padding: const EdgeInsets.all(10.0),

child: AspectRatioVideo(\_controller),

);

}

Widget \_previewImages() {

final Text? retrieveError = \_getRetrieveErrorWidget();

if (retrieveError != null) {

return retrieveError;

}

if (\_mediaFileList != null) {

return Semantics(

label: 'image\_picker\_example\_picked\_images',

child: ListView.builder(

key: UniqueKey(),

itemBuilder: (BuildContext context, int index) {

final String? mime = lookupMimeType(\_mediaFileList![index].path);

// Why network for web?

// See https://pub.dev/packages/image\_picker\_for\_web#limitations-on-the-web-platform

return Semantics(

label: 'image\_picker\_example\_picked\_image',

child: kIsWeb

? Image.network(\_mediaFileList![index].path)

: (mime == null || mime.startsWith('image/')

? Image.file(

File(\_mediaFileList![index].path),

errorBuilder: (BuildContext context, Object error,

StackTrace? stackTrace) {

return const Center(

child:

Text('This image type is not supported'));

},

)

: \_buildInlineVideoPlayer(index)),

);

},

itemCount: \_mediaFileList!.length,

),

);

} else if (\_pickImageError != null) {

return Text(

'Pick image error: $\_pickImageError',

textAlign: TextAlign.center,

);

} else {

return const Text(

'You have not yet picked an image.',

textAlign: TextAlign.center,

);

}

}

Widget \_buildInlineVideoPlayer(int index) {

final VideoPlayerController controller =

VideoPlayerController.file(File(\_mediaFileList![index].path));

const double volume = kIsWeb ? 0.0 : 1.0;

controller.setVolume(volume);

controller.initialize();

controller.setLooping(true);

controller.play();

return Center(child: AspectRatioVideo(controller));

}

Widget \_handlePreview() {

if (isVideo) {

return \_previewVideo();

} else {

return \_previewImages();

}

}

Future<void> retrieveLostData() async {

final LostDataResponse response = await \_picker.retrieveLostData();

if (response.isEmpty) {

return;

}

if (response.file != null) {

if (response.type == RetrieveType.video) {

isVideo = true;

await \_playVideo(response.file);

} else {

isVideo = false;

setState(() {

if (response.files == null) {

\_setImageFileListFromFile(response.file);

} else {

\_mediaFileList = response.files;

}

});

}

} else {

\_retrieveDataError = response.exception!.code;

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(widget.title!),

),

body: Center(

child: !kIsWeb && defaultTargetPlatform == TargetPlatform.android

? FutureBuilder<void>(

future: retrieveLostData(),

builder: (BuildContext context, AsyncSnapshot<void> snapshot) {

switch (snapshot.connectionState) {

case ConnectionState.none:

case ConnectionState.waiting:

return const Text(

'You have not yet picked an image.',

textAlign: TextAlign.center,

);

case ConnectionState.done:

return \_handlePreview();

case ConnectionState.active:

if (snapshot.hasError) {

return Text(

'Pick image/video error: ${snapshot.error}}',

textAlign: TextAlign.center,

);

} else {

return const Text(

'You have not yet picked an image.',

textAlign: TextAlign.center,

);

}

}

},

)

: \_handlePreview(),

),

floatingActionButton: Column(

mainAxisAlignment: MainAxisAlignment.end,

children: <Widget>[

Semantics(

label: 'image\_picker\_example\_from\_gallery',

child: FloatingActionButton(

onPressed: () {

isVideo = false;

\_onImageButtonPressed(ImageSource.gallery, context: context);

},

heroTag: 'image0',

tooltip: 'Pick Image from gallery',

child: const Icon(Icons.photo),

),

),

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

onPressed: () {

isVideo = false;

\_onImageButtonPressed(

ImageSource.gallery,

context: context,

isMultiImage: true,

isMedia: true,

);

},

heroTag: 'multipleMedia',

tooltip: 'Pick Multiple Media from gallery',

child: const Icon(Icons.photo\_library),

),

),

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

onPressed: () {

isVideo = false;

\_onImageButtonPressed(

ImageSource.gallery,

context: context,

isMedia: true,

);

},

heroTag: 'media',

tooltip: 'Pick Single Media from gallery',

child: const Icon(Icons.photo\_library),

),

),

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

onPressed: () {

isVideo = false;

\_onImageButtonPressed(

ImageSource.gallery,

context: context,

isMultiImage: true,

);

},

heroTag: 'image1',

tooltip: 'Pick Multiple Image from gallery',

child: const Icon(Icons.photo\_library),

),

),

if (\_picker.supportsImageSource(ImageSource.camera))

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

onPressed: () {

isVideo = false;

\_onImageButtonPressed(ImageSource.camera, context: context);

},

heroTag: 'image2',

tooltip: 'Take a Photo',

child: const Icon(Icons.camera\_alt),

),

),

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

backgroundColor: Colors.red,

onPressed: () {

isVideo = true;

\_onImageButtonPressed(ImageSource.gallery, context: context);

},

heroTag: 'video0',

tooltip: 'Pick Video from gallery',

child: const Icon(Icons.video\_library),

),

),

if (\_picker.supportsImageSource(ImageSource.camera))

Padding(

padding: const EdgeInsets.only(top: 16.0),

child: FloatingActionButton(

backgroundColor: Colors.red,

onPressed: () {

isVideo = true;

\_onImageButtonPressed(ImageSource.camera, context: context);

},

heroTag: 'video1',

tooltip: 'Take a Video',

child: const Icon(Icons.videocam),

),

),

],

),

);

}

Text? \_getRetrieveErrorWidget() {

if (\_retrieveDataError != null) {

final Text result = Text(\_retrieveDataError!);

\_retrieveDataError = null;

return result;

}

return null;

}

Future<void> \_displayPickImageDialog(

BuildContext context, OnPickImageCallback onPick) async {

return showDialog(

context: context,

builder: (BuildContext context) {

return AlertDialog(

title: const Text('Add optional parameters'),

content: Column(

mainAxisSize: MainAxisSize.min,

children: <Widget>[

TextField(

controller: maxWidthController,

keyboardType:

const TextInputType.numberWithOptions(decimal: true),

decoration: const InputDecoration(

hintText: 'Enter maxWidth if desired'),

),

TextField(

controller: maxHeightController,

keyboardType:

const TextInputType.numberWithOptions(decimal: true),

decoration: const InputDecoration(

hintText: 'Enter maxHeight if desired'),

),

TextField(

controller: qualityController,

keyboardType: TextInputType.number,

decoration: const InputDecoration(

hintText: 'Enter quality if desired'),

),

],

),

actions: <Widget>[

TextButton(

child: const Text('CANCEL'),

onPressed: () {

Navigator.of(context).pop();

},

),

TextButton(

child: const Text('PICK'),

onPressed: () {

final double? width = maxWidthController.text.isNotEmpty

? double.parse(maxWidthController.text)

: null;

final double? height = maxHeightController.text.isNotEmpty

? double.parse(maxHeightController.text)

: null;

final int? quality = qualityController.text.isNotEmpty

? int.parse(qualityController.text)

: null;

onPick(width, height, quality);

Navigator.of(context).pop();

}),

],

);

});

}

}

typedef OnPickImageCallback = void Function(

double? maxWidth, double? maxHeight, int? quality);

class AspectRatioVideo extends StatefulWidget {

const AspectRatioVideo(this.controller, {super.key});

final VideoPlayerController? controller;

@override

AspectRatioVideoState createState() => AspectRatioVideoState();

}

class AspectRatioVideoState extends State<AspectRatioVideo> {

VideoPlayerController? get controller => widget.controller;

bool initialized = false;

void \_onVideoControllerUpdate() {

if (!mounted) {

return;

}

if (initialized != controller!.value.isInitialized) {

initialized = controller!.value.isInitialized;

setState(() {});

}

}

@override

void initState() {

super.initState();

controller!.addListener(\_onVideoControllerUpdate);

}

@override

void dispose() {

controller!.removeListener(\_onVideoControllerUpdate);

super.dispose();

}

@override

Widget build(BuildContext context) {

if (initialized) {

return Center(

child: AspectRatio(

aspectRatio: controller!.value.aspectRatio,

child: VideoPlayer(controller!),

),

);

} else {

return Container();

}

}

}