How to upload to Firebase Storage with Flutter

This is the second part of our tutorial series on how to upload images to Firebase Storage using the Flutter app. Click here for part one.

In this shot, we are going to deal with UI as well as functionality. The idea is to create a simple interface containing an image display container and a button to upload an image after selecting it from the gallery. After an image is selected, the file will be automatically uploaded to the Firebase storage and then displayed on the screen.

Implementing image upload in a Flutter project

In order to use Firebase services and access the device gallery, we need to make use of some library packages. The list of plugins to be included are provided in the code snippet below:

```
firebase_core: ^0.4.5
image_picker: ^0.6.7+14
firebase_storage: ^3.1.6
permission_handler: ^5.0.1+1
```

Here, we have four plugins:

`firebase_core`: This is a core Firebase package that allows us to bind Firebase services to Flutter projects and use the service modules.

`image_picker`: This package offers modules access to the device gallery to select an image.

`firebase_storage`: This package offers Firebase storage service modules to be used in Flutter. The methods offered will allow us to configure the uploading feature.

`permission_handler`: This package allows us to handle the permission to our storage. It provides a simple interface to control the access to a users' device permissions.

We need to copy these plugins, paste them into the `pubspec.yaml` file, and save it to install them. We can also run the following command to get them installed:

```
flutter get packages
```

After installation, we can import the packages to the `main.dart` file as directed in the code snippet below:

```
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:permission_handler/permission_handler.dart';
```

Next, we need to create a new stateful widget class called `ImageUpload` in the `main.dart` file as shown in the code snippet below:

```
class ImageUpload extends StatefulWidget {
  @override
  _ImageUploadState createState() => _ImageUploadState();
}

class _ImageUploadState extends State<ImageUpload> {
  String imageUrl;

  @override
  Widget build(BuildContext context) {
    backgroundColor: Colors.white,
    return Scaffold(
      appBar: AppBar(
        title: Text('Upload Image', style: TextStyle(color: Colors.black87, fontWeight: FontWei
ght.bold),),
        centerTitle: true,
        elevation: 0.0,
        backgroundColor: Colors.white,
      ),
      body: Container()
   );
  }
}
```

In the widget builder function, we have returned a `Scaffold` widget with a `AppBar` and an empty `Container` as a `body` property.

Now, we need to call the class object in the `MyApp` stateless class object as shown below:

```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: ImageUpload(),
    );
  }
}
```

If we run the app using the command `flutter run`, we will get the following result in our device or emulator: *Let’s get started!*

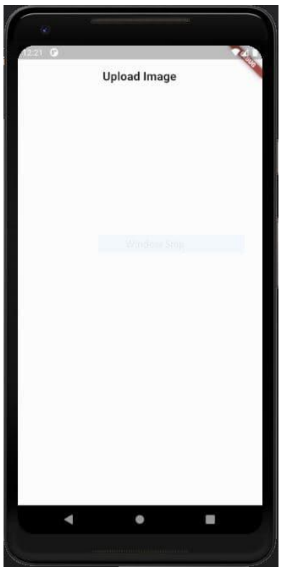


Image placeholder section

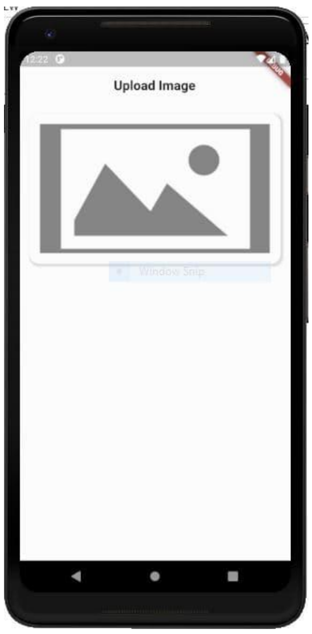
We will implement the body section of the `Scaffold` – it will contain an Image Placeholder section that will display the image after the upload and a button to lead us to the device gallery and upload the image after selection.

First, we are going to implement the Image Placeholder section with the following code:

```
body: Container(
        color: Colors.white,
        child: Column(
          children: <Widget>[
            Container(
              margin: EdgeInsets.all(15),
              padding: EdgeInsets.all(15),
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.all(
                  Radius.circular(15),
                ),
                border: Border.all(color: Colors.white),
                boxShadow: [
                  BoxShadow(
                    color: Colors.black12,
                    offset: Offset(2, 2),
                    spreadRadius: 2,
                    blurRadius: 1,
                  ),
                ],
              ),
              child: (imageUrl != null)
                ? Image.network(imageUrl)
                : Image.network('https://i.imgur.com/sUFH1Aq.png')
            ),
          ],
        ),
      ),
```

Here, we have rendered a placeholder image using the `Image.network` widget, but in your case an actual `imageUrl` will available after the upload.

We will get the following result in our emulator screen:

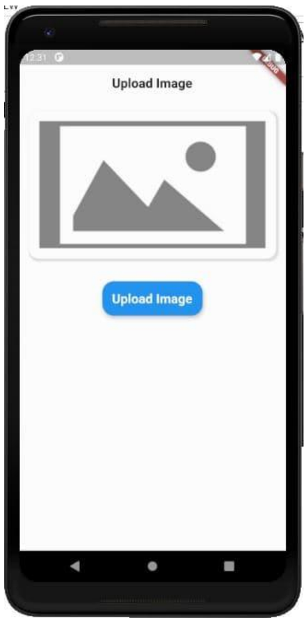


How to add a raised button**

Now, we are going to add the simple button below the Image Placeholder section using the `RaisedButton` widget. Both the placeholder and button are separated by a `SizedBox` widget. We need to place the code from the following code snippet just below the `Container` widget housing the Image Placeholder section:

```
SizedBox(height: 20.0,),
RaisedButton(
  child: Text("Upload Image", style: TextStyle(color: Colors.white, fontWeight: FontWeight.bold
, fontSize: 20)),
  onPressed: (){},
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(18.0),
    side: BorderSide(color: Colors.blue)
  ),
  elevation: 5.0,
  color: Colors.blue,
  textColor: Colors.white,
  padding: EdgeInsets.fromLTRB(15, 15, 15, 15),
  splashColor: Colors.grey,
),
```

We will get the result as shown in the emulator screenshot below:



In the button code, we have not yet implemented any function to perform the upload action; so, we need to devise the function itself.

Function to upload image to Firebase Storage

Here, we are going to implement a function called `uploadImage` that will contain all the configuration from selecting an image from the gallery to uploading it to Firebase Storage to viewing the image in the Image Placeholder section after upload.

First, we need to initialize some plugin instances as directed in the code snippet:

```
uploadImage() async {
    final _firebaseStorage = FirebaseStorage.instance;
    final _imagePicker = ImagePicker();
    PickedFile image;
}
```

Then, we need to get permission to access the device gallery using the `request` method provided by the permission plugin. We are going to store the permission status in the `permissionStatus` variable as shown in the code snippet:

```
await Permission.photos.request();
var permissionStatus = await Permission.photos.status;
```

Next, we need to check if the permission is allowed and then select the image from the gallery. We need to get the `path` file.

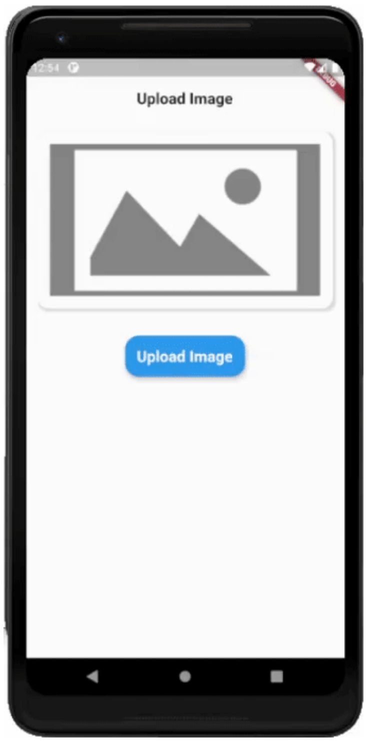
By using the `putFile` method from the `firebaseStorage` package module, we can upload the image to Firebase Storage. The `child` method allows us to set the path and image name to that which will be available in the Firebase storage. The overall implementation of the upload function is provided in the code snippet:

```
uploadImage() async {
    final _firebaseStorage = FirebaseStorage.instance;
    final _imagePicker = ImagePicker();
    PickedFile image;
    //Check Permissions
    await Permission.photos.request();

    var permissionStatus = await Permission.photos.status;

    if (permissionStatus.isGranted){
      //Select Image
      image = await _imagePicker.getImage(source: ImageSource.gallery);
      var file = File(image.path);

      if (image != null){
        //Upload to Firebase
        var snapshot = await _firebaseStorage.ref()
        .child('images/imageName')
        .putFile(file).onComplete;
        var downloadUrl = await snapshot.ref.getDownloadURL();
        setState(() {
          imageUrl = downloadUrl;
        });
      } else {
        print('No Image Path Received');
      }
    } else {
      print('Permission not granted. Try Again with permission access');
    }
  }
```

Then, we simply need to call the `uploadImage` function in the `onPressed` functional property of the `RaisedButton` widget:

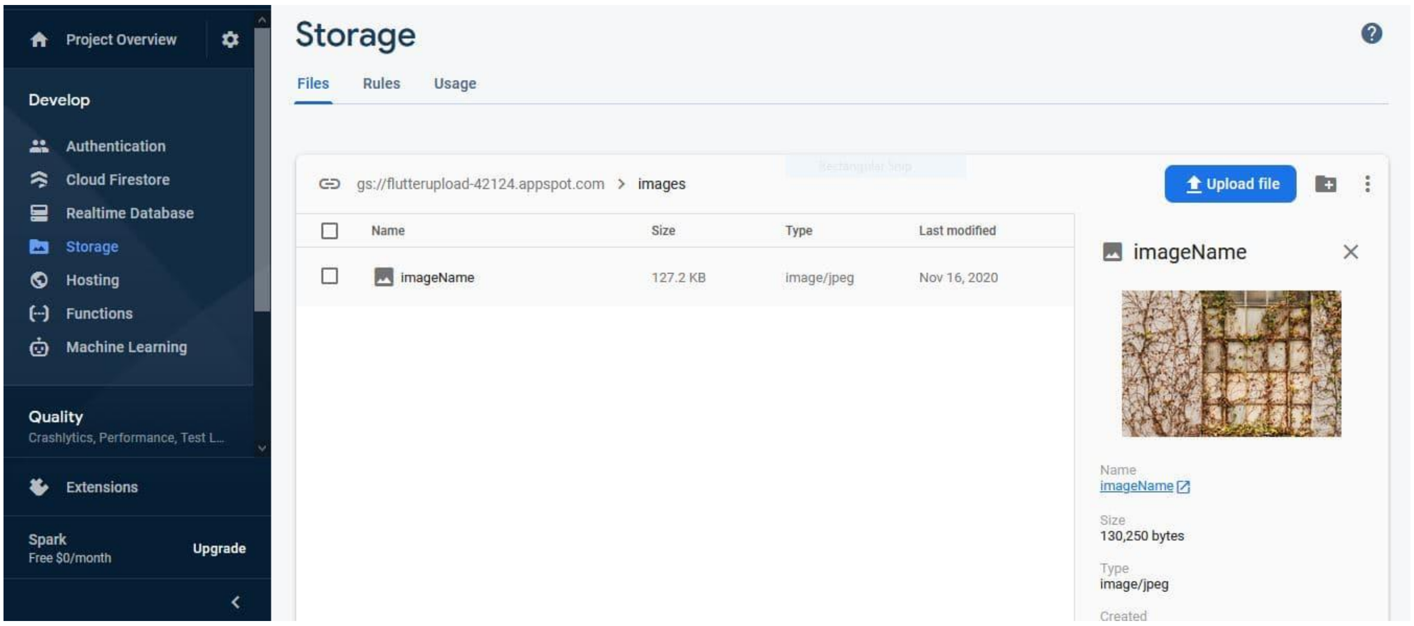
```
RaisedButton(
    child: Text("Upload Image", style: TextStyle(color: Colors.white, fontWeight: FontWeight.bo
ld, fontSize: 20)),
    onPressed: (){
      uploadImage();
    },
```

The final result is shown below:



Note that, it may take a while to upload the image to Firebase storage based on the size of the image file. After the successful upload, the image is displayed in the Image Placeholder section.

The image uploaded to the Firebase Storage will be available in the Firebase Storage console as shown in the screenshot below:



gs://flutterupload-42124.appspot.com > images

Name	Size	Type	Last modified
imageName	127.2 KB	image/jpeg	Nov 16, 2020

imageName

Name
imageName

Size
130,250 bytes

Type
image/jpeg

Created

We have now successfully uploaded the image to Firebase Storage using a Flutter app.

Conclusion

The aim of this tutorial series is to demonstrate steps to upload a file to Firebase Storage service using a Flutter app with dart programming. The focus is not given to creating a sophisticated UI but to the feature itself.

I hope that this step-by-step guidance will help you implement a similar feature in your own Flutter application. The implementation can be applied to other files as well, not just images. Make sure you try it.

Until next time folks, Happy coding!