Pavneet Singh

Integrate Firebase Storage and Image Picker in React Native

Pavneet Singh

Oct 17, 2020 • 12 Min read • 1,362 Views

Oct 17, 2020 • 12 Min read • 1,362 Views

Web Development Front End Web Dev... Client-side Framew... React

Introduction

4

Introduction

The Firebase storage service is used to store media content for all the Firebase apps (Android, iOS, and web) under a linked Firebase project. The Firebase storage service is built upon Google Cloud Storage, which allows web apps to access the content of storage directly. The data is stored in Google Cloud Storage buckets, which are nothing but a namespace to manage data and access controls. The storage service can be added with a separate NPM dependency.

The images can be fetched from the device or camera using the `react-native-image-picker` NPM module.

This guide covers the implementation of Firebase storage and image picker module to fetch and store an image from the device to a Firebase storage bucket. Follow these other guides to complete the implementation:

1. Setting Up a Firebase Project for React Native
2. Integrate Firebase Storage and Image Picker in React Native (current guide)
3. Upload Images to Firebase Storage in React Native

The optimized codebase is available on RNFirebaseStorage repository.

Setting Up Firebase Storage Dependencies

Every Firebase service requires a specific NPM module to communicate with the Firebase server apps. The Firebase storage APIs can be integrated using the `react-native-firebase/storage` NPM module:

bash

```
# Install the storage module
npm install @react-native-firebase/storage
```

```
3
4       # setup iOS project
5       cd ios && pod install --repo-update && cd ..
```

The commands will install the Firebase storage module. Install the iOS `pod` dependency for the Firebase storage.

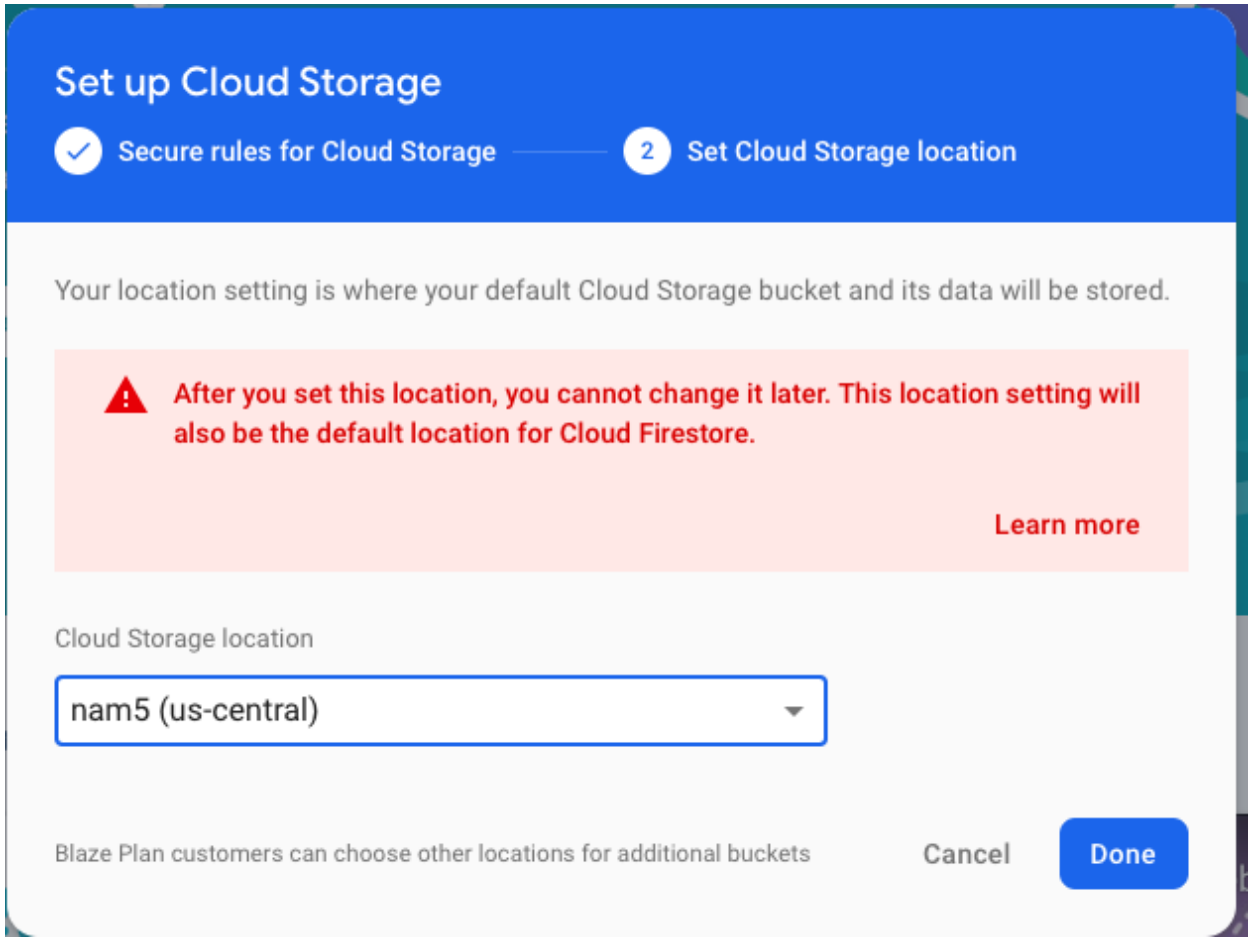
> `@react-native-firebase/app` *NPM modules is required to use most of the Firebase NPM modules.*

Setting Up Firebase Storage in Console

By default, Firebase creates a storage bucket to store the media content and name the bucket using the Firebase project ID as `gs://firebase-project-id.appspot.com`.

Create a default storage bucket in the Firebase project:

1. Select the **Storage** option form the left panel and click on **Get Started**.
2. Continue with the onboarding flow with the **next** option and select any storage location near to your location.



Firebase protects the storage content from unauthorized access by using the `allow read, write: if request.auth != null;` rule.

3. Choose any cloud storage location near to your current location and click **Done**.
4. Go to the **Rules** tab and change the rules for the public access:

```
rules_version = '2';
service firebase.storage {
```

```
    match /b/{bucket}/o {
      match /{allPaths=**} {
        allow read, write;
      }
    }
```

This will enable public access to the Firebase storage bucket to every user without integrating the Firebase authentication service.

> *Do not use the public access rule `allow read, write`; in production, it can cause data protection issues.*

Setting Up Image Picker

The images on Android or iOS devices are managed via different built-in apps (Gallery or Photos), and these images can be accessed via a specific path (URI for iOS) value. The react-native-image-picker module provides React Native-specific APIs to get the path/URI of an image on the underlying native device. Follow the below steps to install and configure the image picker:

Install Image Picker

Execute the below commands to install the image picker module and iOS `pod` dependency

```sh
npm install react-native-image-picker
cd ios && pod install && cd ..
```

Add Permission

Both Android and iOS follow the permission model to inform the user about the resource used by an app, so these permissions should be added in configurations files in the native projects.

- **Add Android Permission**: Add the required permissions in the `RNFirebaseStorage/android/app/src/main/AndroidManifest.xml` file to get the camera and storage access on an android device:

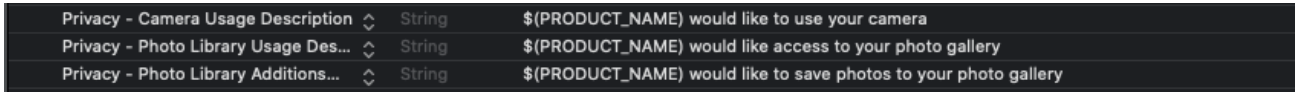
```xml
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/
```

- **Add iOS permissions**: The below permissions (key-string pairs) should be added in the `RNFirebaseStorage/ios/RNFirebaseStorage/info.plist` file to get access to photos, camera, and photo storage. Open the `info.plist` file with any text editor and then copy and paste all three key-string pairs from the below snippet. In Xcode, right-click on the `info.plist` file, choose the **Opens As** option, and then `Source Code` option:

xml

```xml
<plist version="1.0">
  <dict>
    <key>NSPhotoLibraryUsageDescription</key>
    <string>$(PRODUCT_NAME) would like access to your photo gallery</stri
    <key>NSCameraUsageDescription</key>
    <string>$(PRODUCT_NAME) would like to use your camera</string>
    <key>NSPhotoLibraryAddUsageDescription</key>
    <string>$(PRODUCT_NAME) would like to save photos to your photo galle
    </dict>
</plist>
```

Alternately, this can be done in the Xcode UI by using the **+** button to add key and string values:



Optionally, clean the Android and iOS projects:

sh

```sh
#android
cd android && ./gradlew clean && cd ..
#iOS
cd ios && xcodebuild clean && cd ..
```

Implement Image Picker

The image picker can get the image either from the photo gallery or camera. The UI of the image picker dialog can also be customized using the `Options` object that can be passed as the first parameter to `showImagePicker` function call. The second parameter of `showImagePicker` is a callback function that will be called with a `response` object which contains the details about the selected image:

JSX

```
chooseFile = () => {
    this.setState({ status: '' });
    var options = {
        title: 'Select Image',
        storageOptions: {
            skipBackup: true, // do not backup to iCloud
            path: 'images', // store camera images under Pictures/images
        },
    };
    ImagePicker.showImagePicker(options, response => {
        if (response.didCancel) {
            console.log('User cancelled image picker', storage());
        } else if (response.error) {
            console.log('ImagePicker Error: ', response.error);
        } else if (response.customButton) {
            console.log('User tapped custom button: ', response.customBut
        } else {
            let path = this.getPlatformPath(response).value;
        }
    });
};

```

```
    /**
     * Get platform specific value from response
     */
    getPlatformPath({ path, uri }) {
        return Platform.select({
            android: { "value": path },
            ios: { "value": uri }
        })
    }
```

The `storageOptions` object defined the `path` to store clicked images under the `Document/Images` for iOS and `Pictures/images` for Android. A `skipBackup: true` property will instruct iOS to skip the iCloud image backup for camera images.

Here's the complete code to trigger `chooseFile` function to fetch and display the selected image on screen:

jsx

```
// App.js
/**
 * @author Pavneet Singh
 */

import React from "react";
import {
    StyleSheet,
    View,
    Button,
    Image,
```

```
        ActivityIndicator,
        Platform,
        SafeAreaView,
        Text,
    } from "react-native";
    import storage from '@react-native-firebase/storage';
    import ImagePicker from 'react-native-image-picker';
    export default class App extends React.Component {

        state = {
            // placeholder image
            imagePath: require("./img/default.jpg")
        }

        chooseFile = () => {
            var options = {
                title: 'Select Image',
                storageOptions: {
                    skipBackup: true, // do not backup to iCloud
                    path: 'images', // store camera images under Pictures/im
                },
            };
            ImagePicker.showImagePicker(options, response => {
                if (response.didCancel) {
                    console.log('User cancelled image picker', storage());
                } else if (response.error) {
                    console.log('ImagePicker Error: ', response.error);
                } else if (response.customButton) {
                    console.log('User tapped custom button: ', response.cust
                } else {
                    let path = this.getPlatformPath(response).value;
                    let fileName = this.getFileName(response.fileName, path)
                    this.setState({ imagePath: path });
                }
            });
```

```
47          };
48
49          /**
50           * Get the file name and handle the invalid null case
51           */
52          getFileName(name, path) {
53              if (name != null) { return name; }
54
55              if (Platform.OS === "ios") {
56                  path = "~" + path.substring(path.indexOf("/Documents"));
57              }
58              return path.split("/").pop();
59          }
60
61          /**
62           * Get platform specific value from response
63           */
64          getPlatformPath({ path, uri }) {
65              return Platform.select({
66                  android: { "value": path },
67                  ios: { "value": uri }
68              })
69          }
70
71          /**
72           * Get platform-specific Uri with the required format
73           */
74          getPlatformURI(imagePath) {
75              let imgSource = imagePath;
76              if (isNaN(imagePath)) {
77                  imgSource = { uri: this.state.imagePath };
78                  if (Platform.OS == 'android') {
79                      imgSource.uri = "file:///" + imgSource.uri;
80                  }
81              }
```

```
            return imgSource
        }

        render() {
            let { imagePath } = this.state;
            let imgSource = this.getPlatformURI(imagePath)
            return (
                <SafeAreaView style={styles.container}>
                    <View style={styles.imgContainer}>
                        <Image style={styles.uploadImage} source={imgSource}
                        <View style={styles.eightyWidthStyle} >
                            <Button title={'Upload Image'} onPress={this.cho
                        </View>
                    </View>
                </SafeAreaView>
            )
        }
    }

    const styles = StyleSheet.create({
        container: {
            flex: 1,
            width: '100%',
            height: '100%',
            backgroundColor: '#e6e6fa',
        },
        imgContainer: {
            alignItems: 'center',
            justifyContent: 'center',
            position: 'absolute',
            width: '100%',
            height: '100%'
        },
        eightyWidthStyle: {
            width: '80%',
```

```
            margin: 2,
        },
        uploadImage: {
            width: '80%',
            height: 300,
        },
    });
```

A successful image selection will trigger the `else` case in `chooseFile` method and provide the image details via the `response` object. The `getPlatformPath` function will return a platform-specific path/URI value that is being stored in the `state` object to update the image on the screen.

Conclusion

Firebase services like storage, database, and Firestore can use the Firebase rules to validate the data and users. Firebase storage is built upon the Google Cloud Storage service so it can store petabytes of data. The `image-picker` module offers a quick way to fetch images from the devices. It also supports functions like `launchCamera` and `launchImageLibrary` to use a specific image

source. Follow this guide to learn about uploading a selected image to the Firebase storage bucket. Happy coding!