

Tips



Pavneet Singh

- [Introduction](#)
- [Uploading Files to Firebase Storage](#)
- [Image Picker Optimizations](#)
- [Display Uploading Status](#)
- [Tips](#)
- [Conclusion](#)
- [Top](#) ^



# Upload Images to Firebase Storage in React Native

Pavneet Singh

Oct 17, 2020 • 10 Min read • 4,695 Views

Oct 17, 2020 • 10 Min read • 4,695 Views

Web Development

Front End Web Dev...

Client-side Framew...

React

## Introduction

The Firebase storage NPM module provides access to the Firebase app instance to create a reference object that is used to upload images to the Firebase storage. The Firebase storage bucket can store the data using a file or folder. A `reference` instance can also take a string path value to upload an image to Firebase storage.

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimizations](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

This guide explains the steps to upload a selected image from a device to Firebase storage using the image picker helper functions.



Check out these previous guides for more context and complete details:

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

## Uploading Files to Firebase Storage

The `@react-native-firebase/storage` module provides access to the `FirebaseApp` instance that is used to upload media content into the Firebase storage. A `storage()` instance contains all the necessary details to verify and establish a connection with the Firebase server app to access storage contents.

Follow these steps to upload an image to Firebase storage:

1. Import the default Firebase app module.
2. Create an instance of the Firebase storage API and use the `ref` function to get a reference instance to upload an image.
3. Invoke the `putFile` function with a file path/URI to upload the file to Firebase storage.

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

4. The `puFile` function returns a `task` object that supports `Promise` based implementation to track the result of the upload process using the `then` function.



JSX

```
import storage from '@react-native-firebase/storage'; // 1

1      uploadImageToStorage(path, imageName) {
2          let reference = storage().ref(imageName);           // 2
3          let task = reference.putFile(path);                  // 3
4
5          task.then(() => {                                     // 4
6              console.log('Image uploaded to the bucket!');
7          }).catch((e) => console.log('uploading image error => ', e));
8      }
9
10
```

The `reference` instance also supports different file formats, such as `put(Blob | Uint8Array | ArrayBuffer)`, and provides a `putString` to upload different types of data.

## Image Picker Optimizations

The image picker implementation can be optimized with the `Platform.select` function. A `response` object can be used to retrieve platform-specific values and the name of the file:

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)



- **Getting a Platform-specific Image Path/URI:** Android and iOS use different path/URI schema formats to access the images. iOS uses the `file:///` schema in path value, whereas Android uses a plain URI to access images. The different file path or URI values can be handled using the `Platform` API to verify the OS type at run time:

JSX

```
1  /**
2   * Get platform specific value from response
3   */
4  getPlatformPath({ path, uri }) {
5      return Platform.select({
6          android: { path },
7          ios: { uri }
8      })
9  }
```

**Note:** It's recommended to use platform APIs as any change in API can break our custom logic. Also, the image path should be formatted to support the `file:///` protocol.

- **Getting an Image Name:** The `response` object provides the name of the file using the `response.fileName` property, which can be used to store the images with default names instead of custom names.

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

# Display Uploading Status

It is important to inform the user about the image loading process. This can be done using the `ActivityIndicator` component to show a circular progress indicator on the UI. The progress indicator is shown during the image uploading process. Follow these steps to implement it:

1. Create an `isLoading` property to track the uploading task status.
2. Update the value of `isLoading` to `true` at the beginning of the uploading task and set it to `false` inside the `then` and `catch`.
3. Import `ActivityIndicator` and display it using the `isLoading` value:

JSX

```
1 // App.js
2 /**
3  * @author Pavneet Singh
4  */
5
6 import React from "react";
7 import {
8   StyleSheet,
9   View,
10  Button,
11  Image,
12  ActivityIndicator,
13  Platform,
14  SafeAreaView,
15  Text,
16 } from "react-native";
```

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

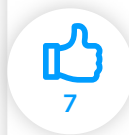


```
17 import storage from '@react-native-firebase/storage';
18 import ImagePicker from 'react-native-image-picker';
19 export default class App extends React.Component {
20
21   state = {
22     imagePath: require("./img/default.jpg"),
23     isLoading: false,
24     status: '',
25   }
26
27   chooseFile = () => {
28     this.setState({ status: '' });
29     var options = {
30       title: 'Select Image',
31       customButtons: [
32         { name: 'customOptionKey', title: 'Choose Photo from Cus
33       ],
34       storageOptions: {
35         skipBackup: true, // do not backup to iCloud
36         path: 'images', // store camera images under Pictures/in
37       },
38     };
39     ImagePicker.showImagePicker(options, response => {
40       if (response.didCancel) {
41         console.log('User cancelled image picker', storage());
42       } else if (response.error) {
43         console.log('ImagePicker Error: ', response.error);
44       } else if (response.customButton) {
45         console.log('User tapped custom button: ', response.cust
46       } else {
47         let path = this.getPlatformPath(response).value;
48         let fileName = this.getFileName(response.fileName, path)
49         this.setState({ imagePath: path });
50         this.uploadImageToStorage(path, fileName);
51       }
52     });
53   }
54 }
```

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

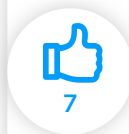


```
52     });
53   };
54
55   getFileName(name, path) {
56     if (name !== null) { return name; }
57
58     if (Platform.OS === "ios") {
59       path = "~" + path.substring(path.indexOf("/Documents"));
60     }
61     return path.split("/").pop();
62   }
63
64   uploadImageToStorage(path, name) {
65     this.setState({ isLoading: true });
66     let reference = storage().ref(name);
67     let task = reference.putFile(path);
68     task.then(() => {
69       console.log('Image uploaded to the bucket!');
70       this.setState({ isLoading: false, status: 'Image uploaded su
71     }).catch((e) => {
72       status = 'Something went wrong';
73       console.log('uploading image error => ', e);
74       this.setState({ isLoading: false, status: 'Something went wr
75     });
76   }
77
78   /**
79    * Get platform specific value from response
80    */
81   getPlatformPath({ path, uri }) {
82     return Platform.select({
83       android: { "value": path },
84       ios: { "value": uri }
85     })
86   }
```

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)



```
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121

getPlatformURI(imagePath) {
  let imgSource = imagePath;
  if (isNaN(imagePath)) {
    imgSource = { uri: this.state.imagePath };
    if (Platform.OS == 'android') {
      imgSource.uri = "file:/// " + imgSource.uri;
    }
  }
  return imgSource
}

render() {
  let { imagePath } = this.state;
  let imgSource = this.getPlatformURI(imagePath)
  return (
    <SafeAreaView style={styles.container}>
      {this.state.isLoading && <ActivityIndicator size="large"
    <View style={styles.imgContainer}>
      <Text style={styles.boldTextStyle}>{this.state.statu
      <Image style={styles.uploadImage} source={imgSource}
      <View style={styles.eightyWidthStyle} >
        <Button title={'Upload Image'} onPress={this.chc
      </View>
    </View>
  </SafeAreaView>
  )
}
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    width: '100%',
    height: '100%',
```



## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)



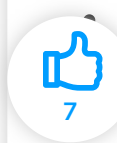
```
122         backgroundColor: '#e6e6fa',
123     },
124     imgContainer: {
125         alignItems: 'center',
126         justifyContent: 'center',
127         position: 'absolute',
128         width: '100%',
129         height: '100%'
130     },
131     eightyWidthStyle: {
132         width: '80%',
133         margin: 2,
134     },
135     uploadImage: {
136         width: '80%',
137         height: 300,
138     },
139     loadingIndicator: {
140         zIndex: 5,
141         width: '100%',
142         height: '100%',
143     },
144     boldTextStyle: {
145         fontWeight: 'bold',
146         fontSize: 22,
147         color: '#5EB0E5',
148     }
149
150     });
```

## Tips



- [Introduction](#)
- [Uploading Files to Fire](#)
- [Image Picker Optimiza](#)
- [Display Uploading Stat](#)
- [Tips](#)
- [Conclusion](#)
- [Top ^](#)

## Tips



- Validating network availability can help to prompt the user to enable internet connectivity for the image loading task. The `react-native-netinfo` NPM module is the official way to identify the network status.
- Use the Firebase authentication feature to allow only authenticated users to access data.
- The `react-native-image-picker` also supports custom buttons with the `customButtons` option property to fetch images from other platforms or custom social media APIs.
- The Firebase rules can also be modified to allow only public read and authenticated writes:

```
1      alternately
2      allow read;
3      allow write: if request.auth != null;
```

Read more about the [Firebase rules here](#).

## Conclusion

Firebase is specifically optimized to address issues on mobile devices with many other great services. It is important to integrate Firebase configuration and rules (or authentication services)

Tips



- [Introduction](#)
- [Uploading Files to Firebase Storage](#)
- [Image Picker Optimizations](#)
- [Display Uploading Status](#)
- [Tips](#)
- [Conclusion](#)
- [Top](#) ^

carefully. React Native offers all the required NPM plugins to integrate Firebase services and required features to fetch images, check network status, and much more. The complete optimized Firebase is available on the [RNFirestoreStorage](#) repository. Happy coding!