Tips

7

Payneet Singh

Upload Images to Firebase Storage in React Native

Payneet 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.

This guide explains the steps to upload a selected image from a device to Firebase storage using the image picker helper functions. eck out these previous guides for more context and complete tails:

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

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

    uploadImageToStorage(path, imageName) {
        let reference = storage().ref(imageName);          // 2
        let task = reference.putFile(path);                // 3

        task.then(() => {                                  // 4
            console.log('Image uploaded to the bucket!');
        }).catch((e) => console.log('uploading image error => ', e));
    }
```

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

7

- **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

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

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.

Display Uploading Status

7

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

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

import React from "react";
import {
    StyleSheet,
    View,
    Button,
    Image,
    ActivityIndicator,
    Platform,
    SafeAreaView,
    Text,
} from "react-native";
```

Tips

7

7

```
import storage from '@react-native-firebase/storage';
import ImagePicker from 'react-native-image-picker';
export default class App extends React.Component {

    state = {
        imagePath: require("./img/default.jpg"),
        isLoading: false,
        status: '',
    }

    chooseFile = () => {
        this.setState({ status: '' });
        var options = {
            title: 'Select Image',
            customButtons: [
                { name: 'customOptionKey', title: 'Choose Photo from Cus
            ],
            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 });
                this.uploadImageToStorage(path, fileName);
            }
```

Tips

7

7

```
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

7

7

```
87
88          getPlatformURI(imagePath) {
89              let imgSource = imagePath;
90              if (isNaN(imagePath)) {
91                  imgSource = { uri: this.state.imagePath };
92                  if (Platform.OS == 'android') {
93                      imgSource.uri = "file:///" + imgSource.uri;
94                  }
95              }
96              return imgSource
97          }
98
99          render() {
100             let { imagePath } = this.state;
101             let imgSource = this.getPlatformURI(imagePath)
102             return (
103                 <SafeAreaView style={styles.container}>
104                     {this.state.isLoading && <ActivityIndicator size="large"
105                     <View style={styles.imgContainer}>
106                         <Text style={styles.boldTextStyle}>{this.state.statu
107                         <Image style={styles.uploadImage} source={imgSource}
108                         <View style={styles.eightyWidthStyle} >
109                             <Button title={'Upload Image'} onPress={this.cho
110                         </View>
111                     </View>
112                 </SafeAreaView>
113             )
114         }
115     }
116
117     const styles = StyleSheet.create({
118         container: {
119             flex: 1,
120             width: '100%',
121             height: '100%',
```

Tips


```
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

7

- 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)

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 codebase is available on the RNFirebaseStorage repository. Happy coding!

Tips

7

7