Name	Shubham Golwal
UID no.	20203000**15**
Experiment No.	10

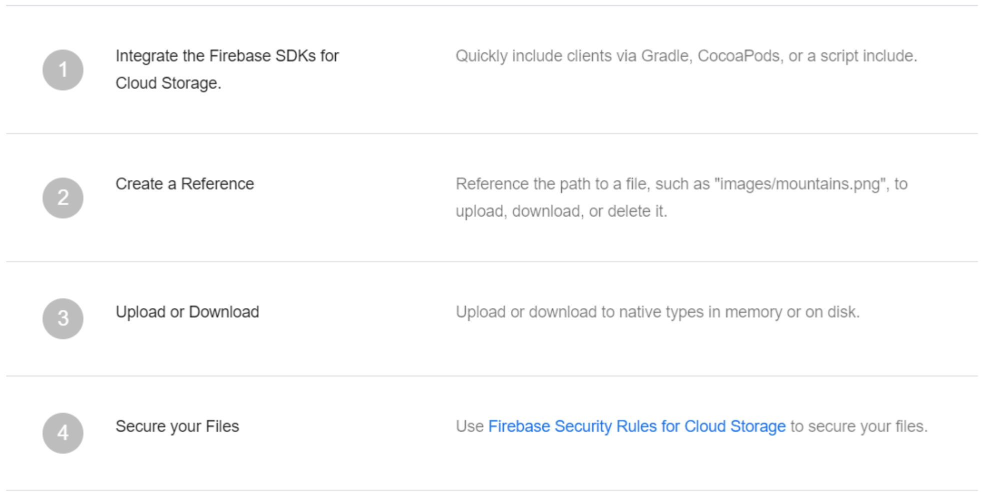
	EXPERIMENT 10
Aim	To develop a multimedia database
Problem Statement	Create a database to store any type of multimedia file and retrieve it back on demand. That database will act as organised storage of your data.
Theory	**What Is Firebase Storage?** Firebase is a Backend-as-a-Service (BaaS) platform that provides developers with a suite of tools and services for building and managing mobile and web applications. It was originally developed by Firebase, Inc. in 2011 and later acquired by Google in 2014. Firebase offers a range of features such as Realtime Database, Cloud Firestore, Cloud Functions, Authentication, Cloud Storage, Hosting, and more. These features enable developers to build scalable and secure applications without worrying about server infrastructure or backend coding. Firebase also provides a comprehensive set of SDKs and APIs for various programming languages and platforms, making it easy for developers to integrate Firebase into their applications. One of the main advantages of Firebase is that it provides real-time data synchronization between the client and the server. This means that any changes made on the client-side are immediately reflected on the server-side and vice versa. Firebase also provides robust security features, including user authentication, access control, and data encryption, to ensure that user data is always secure. **How Does Cloud Storage Fit into the Firebase Architecture?** The Firebase Architecture is made of several components, which work together to provide a framework and infrastructure for developing and hosting web and mobile applications.

- **Firebase Cloud Storage** also called Firebase File Storage or Firebase Object Storage, is an object storage service offered on Google Cloud Platform. When Google Cloud Storage is incorporated into Firebase apps, you gain access to Google security measures and the ability to secure any uploads or downloads in your app. Through the SDK you can also manage your media and access it directly from your storage account. Integration is supported for Android, C++, iOS, Unity, and Web apps.
- **Cloud Fire store** a scalable, flexible database service for server, web, and mobile development. It serves as a NoSQL document database. You can use it to store, query, and sync your app data. It is supported for Android, iOS, and web applications.
- **Authentication** included in the SDK in the form of UI libraries. You can use it as a backend service to authenticate app users via password, phone numbers, or federated ID. Federated options include Google, Twitter, and Facebook. It is supported for Android, iOS, and web applications.
- **Hosting** provides production-grade hosting for your web content. With this component you can deploy apps and serve dynamic or static content via a global content delivery network (CDN). You can also combine the hosting component with Cloud Functions to develop and host microservices. Only web applications are supported.
- **Cloud Functions** enables you to automate code in your backend in response to event triggers. Events can be triggered by HTTPS requests or Firebase features. Cloud Functions are supported for Android, C++, Unity, iOS, and web applications.

How does it work?

Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client can retry the operation right where it left off, saving your users time and bandwidth.

Cloud Storage for Firebase stores your files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows you the flexibility to upload and download files from mobile clients via the Firebase SDKs for Cloud Storage. In addition, you can do server-side processing such as image filtering or video transcoding using the Google Cloud Storage APIs. Cloud Storage scales automatically, meaning

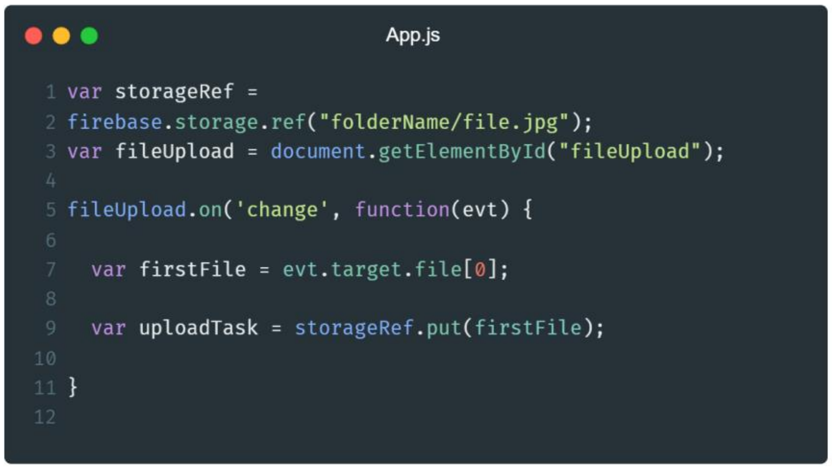
	that there is no need to migrate to any other provider. Learn more about all the benefits of our integration with Google Cloud. The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want. **Implementation path**  1 Integrate the Firebase SDKs for Cloud Storage. — Quickly include clients via Gradle, CocoaPods, or a script include. 2 Create a Reference — Reference the path to a file, such as "images/mountains.png", to upload, download, or delete it. 3 Upload or Download — Upload or download to native types in memory or on disk. 4 Secure your Files — Use Firebase Security Rules for Cloud Storage to secure your files. **Firebase Storage Methods** When using Firebase storage, several methods can help you simplify management, monitor your status, and improve your security. **Accessing Files Through References** Although Firebase Storage uses a real-time database you can access data via a familiar file/folder system. When accessing files, you simply need to call the reference where your file is stored. References also enable you to control where files are stored and how files are labelled. To access files, you can use the following code (for web applications, similar code is available for Android or iOS via the relevant SDKs):

var storageRef = firebase.storage.ref("folderName/file.jpg");

Uploading Files

Uploading files is easy if you know the reference for where you want your file stored. With web apps, you can upload files via an <input type=" file" /> element. When uploading files from Android and iOS, you can upload files from memory or from a stream.

To configure file uploads, you can use the following code:

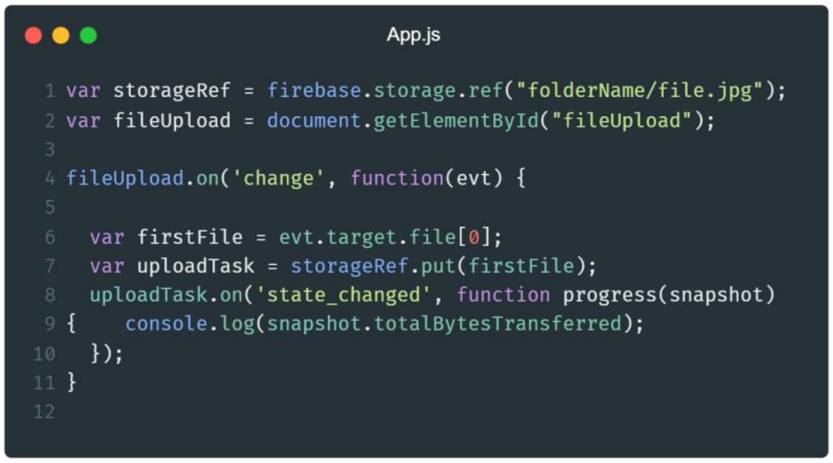


```
var storageRef =
firebase.storage.ref("folderName/file.jpg");
var fileUpload = document.getElementById("fileUpload");

fileUpload.on('change', function(evt) {

  var firstFile = evt.target.file[0];

  var uploadTask = storageRef.put(firstFile);

}

```

Monitor Progress with Tasks

Using Firebase Storage's built-in UploadTask or DownloadTasks listeners you can monitor the progress of your file transfers. These listeners provide a snapshot of your file status, including total file size and how much data has already been transferred. You can then use this information to calculate the percentage completed or to update your app UI.

To configure monitoring, you can use the following code, which is an example specific to file uploads.

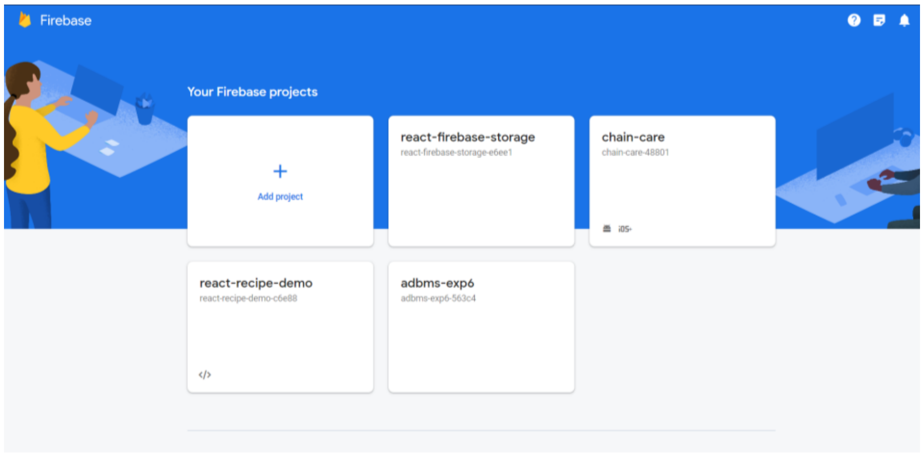


```
var storageRef = firebase.storage.ref("folderName/file.jpg");
var fileUpload = document.getElementById("fileUpload");

fileUpload.on('change', function(evt) {

  var firstFile = evt.target.file[0];
  var uploadTask = storageRef.put(firstFile);
  uploadTask.on('state_changed', function progress(snapshot)
{     console.log(snapshot.totalBytesTransferred);
  });
}
```

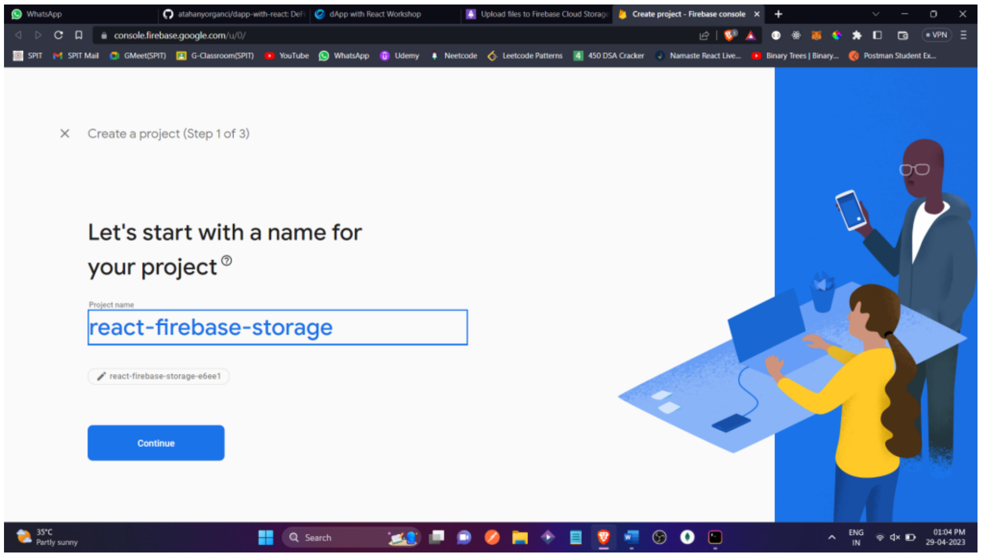
Implementation

1) Create a project on firebase

Go to Firebase Console at https://console.firebase.google.com/ . You will see the homepage:



Click on the Add Project button. Type in the name of your project. I will name mine react-firebase-storage. Accept the Firebase terms and click **Continue**:


Create a project (Step 1 of 3)

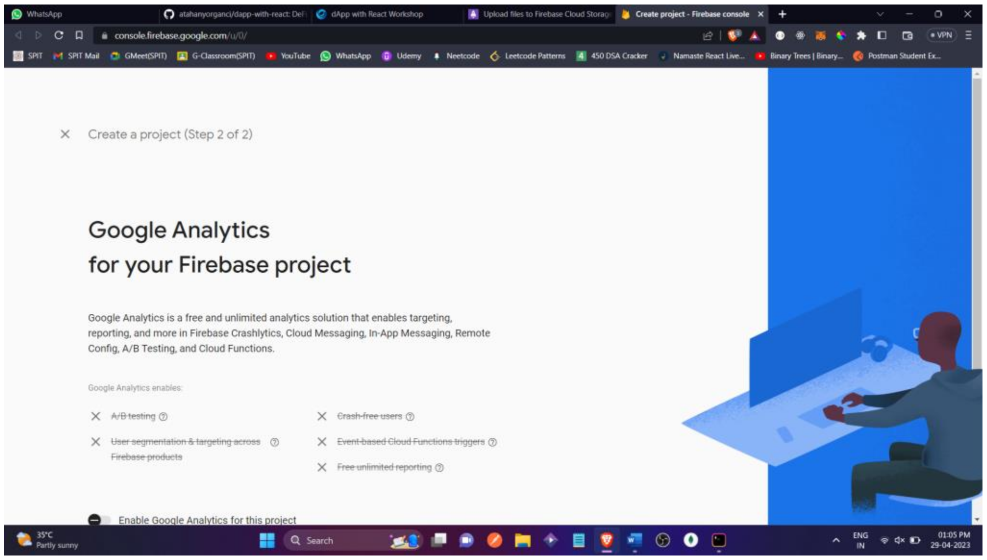
Let's start with a name for your project

Project name

react-firebase-storage

react-firebase-storage-e6ee1

If you would like to use Google Analytics in your project, then leave the Enable Google Analytics toggle on. I don’t need it for this demo, so I’m going to turn it off. Click on Create project and wait for the project to be created:


Create a project (Step 2 of 2)

Google Analytics for your Firebase project

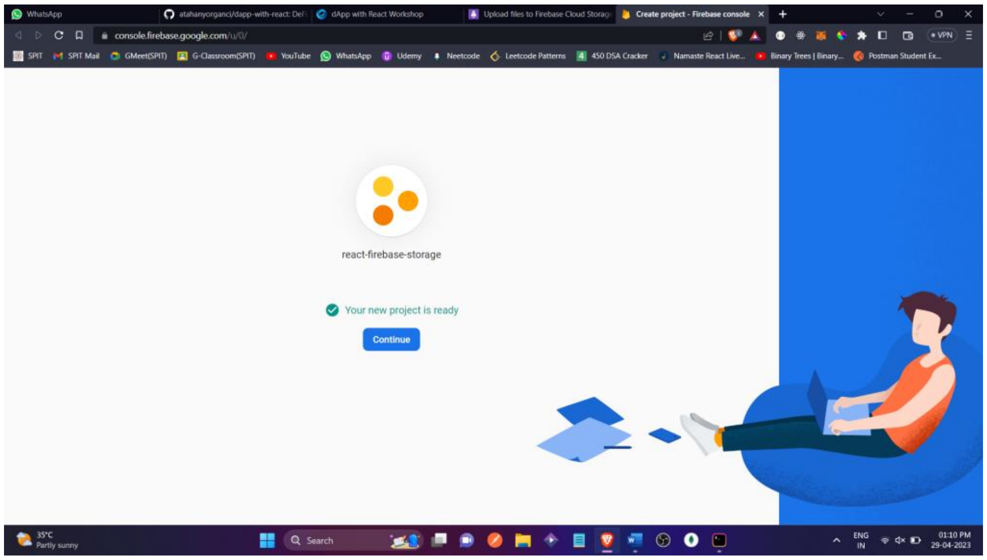
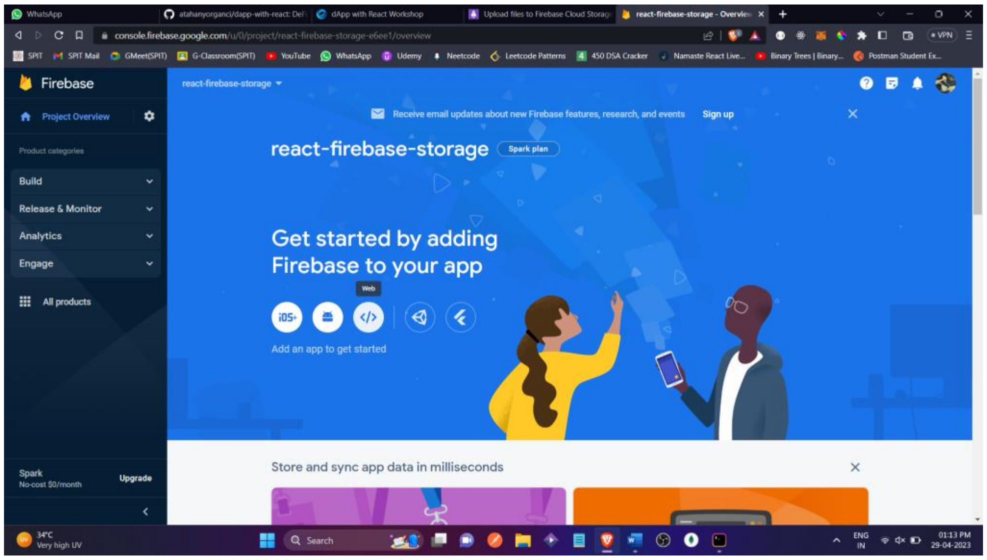
Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, and Cloud Functions.

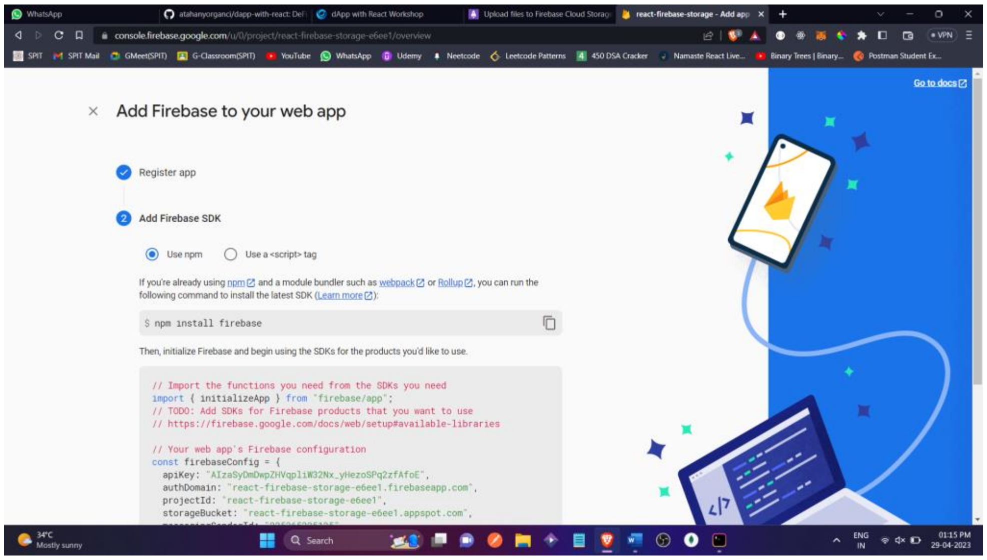
Google Analytics enables:

- ~~A/B testing~~
- ~~User segmentation & targeting across Firebase products~~
- ~~Crash-free users~~
- ~~Event-based Cloud Functions triggers~~
- ~~Free unlimited reporting~~

Enable Google Analytics for this project

Click on Continue to continue to the console:

	 In the next interface, we will select the platform we want to use to build the application we just created. In this case, it's going to be on web, so we choose **web**:  Next, we enter a name to register the app. Since I am not going to host the app on Firebase, I will skip that and click on Register app:



Add Firebase to your web app

Register app

Add Firebase SDK

[✓ Use npm] Use a <script> tag

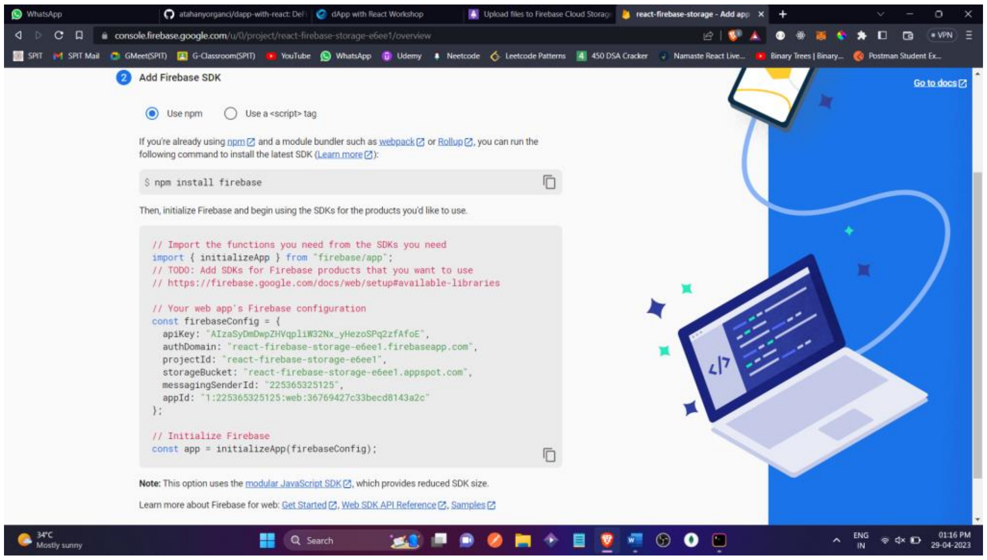
If you're already using npm and a module bundler such as webpack or Rollup, you can run the following command to install the latest SDK (Learn more):

```
$ npm install firebase
```

Then, initialize Firebase and begin using the SDKs for the products you'd like to use.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDmDwpZHVqpliW32Nx_yHezoSPq2zfAfoE",
  authDomain: "react-firebase-storage-e6ee1.firebaseapp.com",
  projectId: "react-firebase-storage-e6ee1",
  storageBucket: "react-firebase-storage-e6ee1.appspot.com",
```

Next, we will initialize a new React app and add Firebase to the project with the credentials provided:



Add Firebase SDK

[✓ Use npm] Use a <script> tag

If you're already using npm and a module bundler such as webpack or Rollup, you can run the following command to install the latest SDK (Learn more):

```
$ npm install firebase
```

Then, initialize Firebase and begin using the SDKs for the products you'd like to use.

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDmDwpZHVqpliW32Nx_yHezoSPq2zfAfoE",
  authDomain: "react-firebase-storage-e6ee1.firebaseapp.com",
  projectId: "react-firebase-storage-e6ee1",
  storageBucket: "react-firebase-storage-e6ee1.appspot.com",
  messagingSenderId: "225365325125",
  appId: "1:225365325125:web:36769427c33becd8143a2c"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Note: This option uses the modular JavaScript SDK, which provides reduced SDK size.

Learn more about Firebase for web: Get Started, Web SDK API Reference, Samples

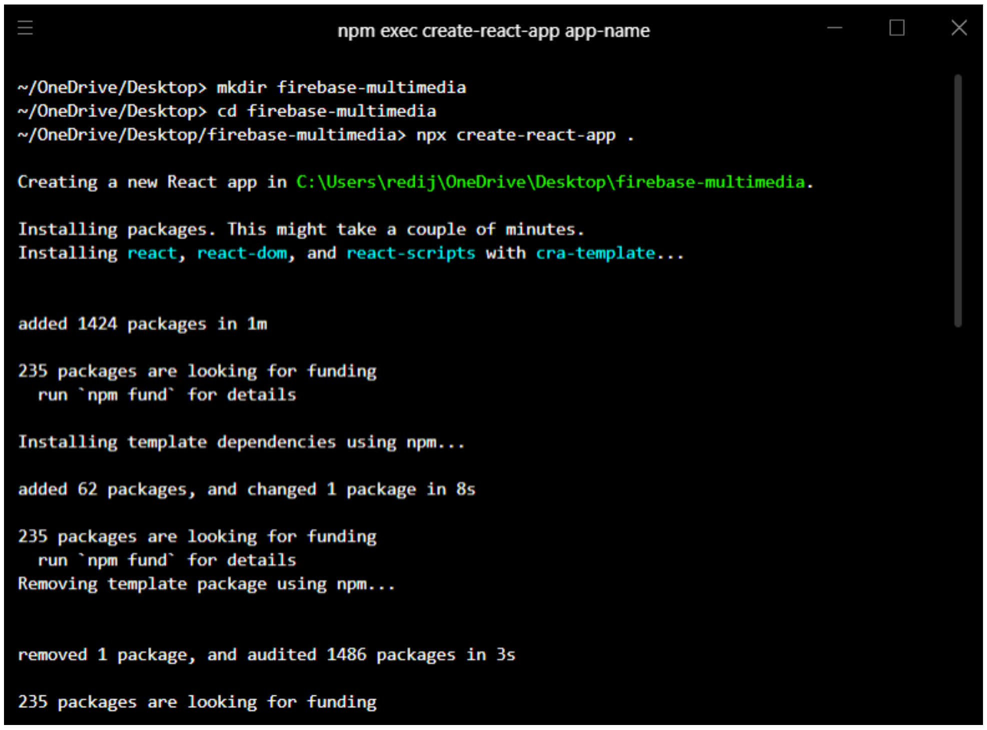
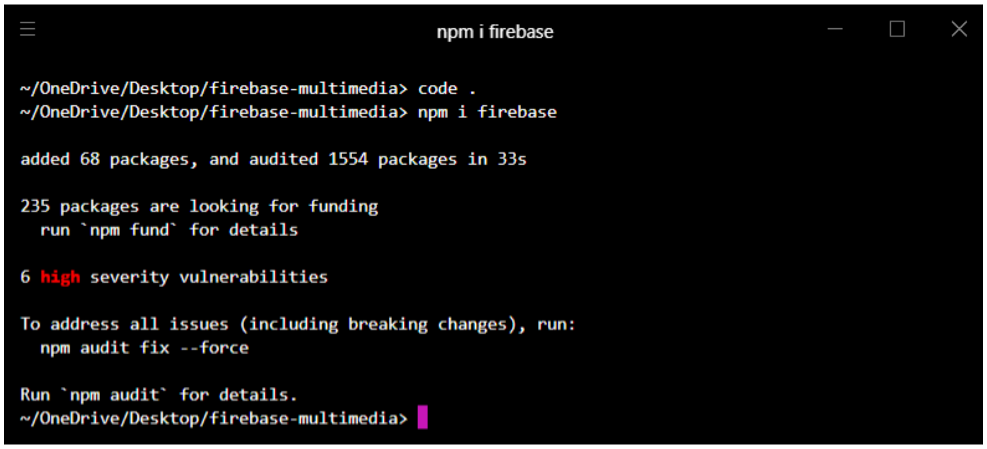
2) Create a new React app

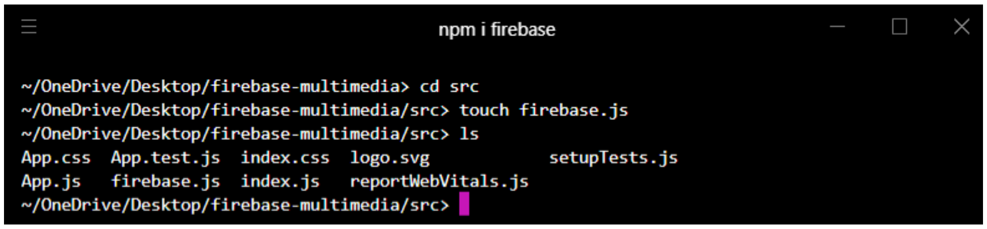
Create a new React app with the command below:

```
npx create-react-app app-name
```

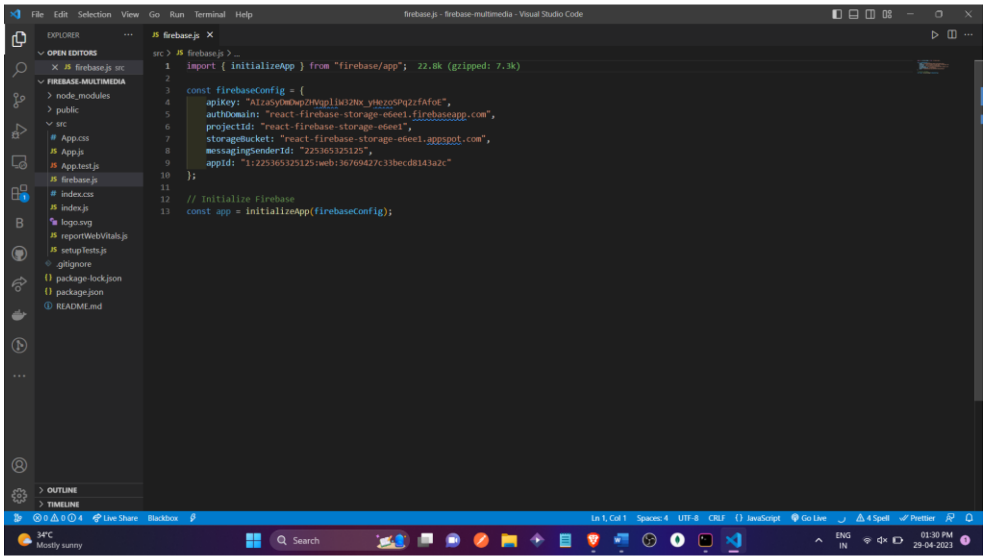
Next, install Firebase as follows:

```
npm install firebase
```

	 npm exec create-react-app app-name
	```
	~/OneDrive/Desktop> mkdir firebase-multimedia
	~/OneDrive/Desktop> cd firebase-multimedia
	~/OneDrive/Desktop/firebase-multimedia> npx create-react-app .

	Creating a new React app in C:\Users\redij\OneDrive\Desktop\firebase-multimedia.

	Installing packages. This might take a couple of minutes.
	Installing react, react-dom, and react-scripts with cra-template...


	added 1424 packages in 1m

	235 packages are looking for funding
	  run `npm fund` for details

	Installing template dependencies using npm...

	added 62 packages, and changed 1 package in 8s

	235 packages are looking for funding
	  run `npm fund` for details
	Removing template package using npm...


	removed 1 package, and audited 1486 packages in 3s

	235 packages are looking for funding
	```
	 npm i firebase
	```
	~/OneDrive/Desktop/firebase-multimedia> code .
	~/OneDrive/Desktop/firebase-multimedia> npm i firebase

	added 68 packages, and audited 1554 packages in 33s

	235 packages are looking for funding
	  run `npm fund` for details

	6 high severity vulnerabilities

	To address all issues (including breaking changes), run:
	  npm audit fix --force

	Run `npm audit` for details.
	~/OneDrive/Desktop/firebase-multimedia>
	```
	Create a new file in the src folder called **firebase.js**. Copy the configuration code from when we created a Firebase project and paste it in the **firebase.js** file.
	Initialize the Firebase app using the config object containing the credentials and export it. You will also export a reference to the storage service, which is used to create references in your storage:

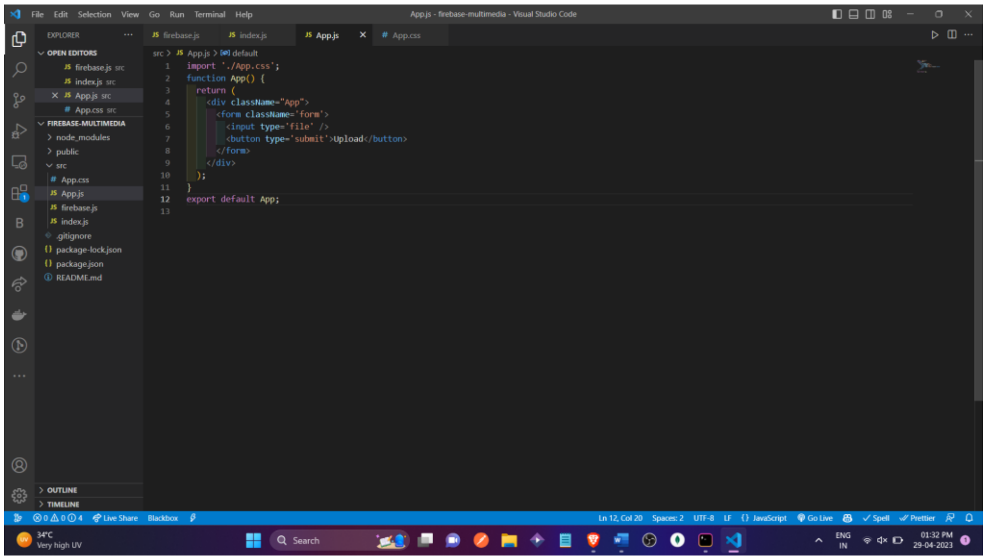


```
~/OneDrive/Desktop/firebase-multimedia> cd src
~/OneDrive/Desktop/firebase-multimedia/src> touch firebase.js
~/OneDrive/Desktop/firebase-multimedia/src> ls
App.css  App.test.js  index.css  logo.svg          setupTests.js
App.js   firebase.js  index.js   reportWebVitals.js
~/OneDrive/Desktop/firebase-multimedia/src>
```

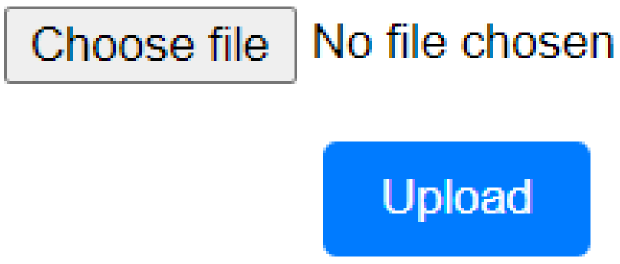


```
import { initializeApp } from "firebase/app";  22.8k (gzipped: 7.3k)

const firebaseConfig = {
    apiKey: "AIzaSyDmDwpZHVqpliW32Nx_yHezoSPq2zfAfoE",
    authDomain: "react-firebase-storage-e6ee1.firebaseapp.com",
    projectId: "react-firebase-storage-e6ee1",
    storageBucket: "react-firebase-storage-e6ee1.appspot.com",
    messagingSenderId: "225365325125",
    appId: "1:225365325125:web:36769427c33becd8143a2c"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

In **App.js**, let us create a form for uploading files and a button for submitting:

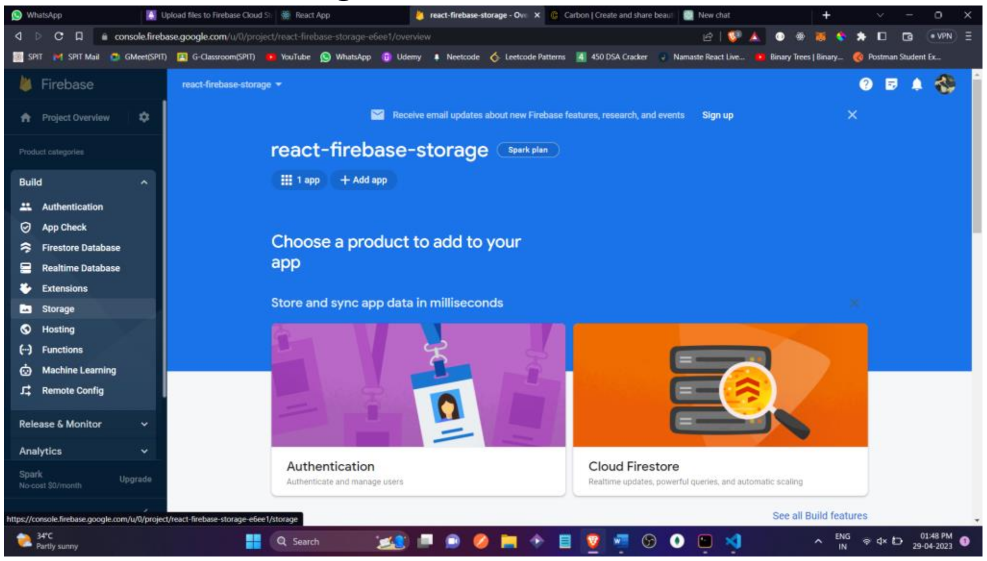


```
import './App.css';
function App() {
  return (
    <div className="App">
      <form className='form'>
        <input type='file' />
        <button type='submit'>Upload</button>
      </form>
    </div>
  );
}
export default App;
```

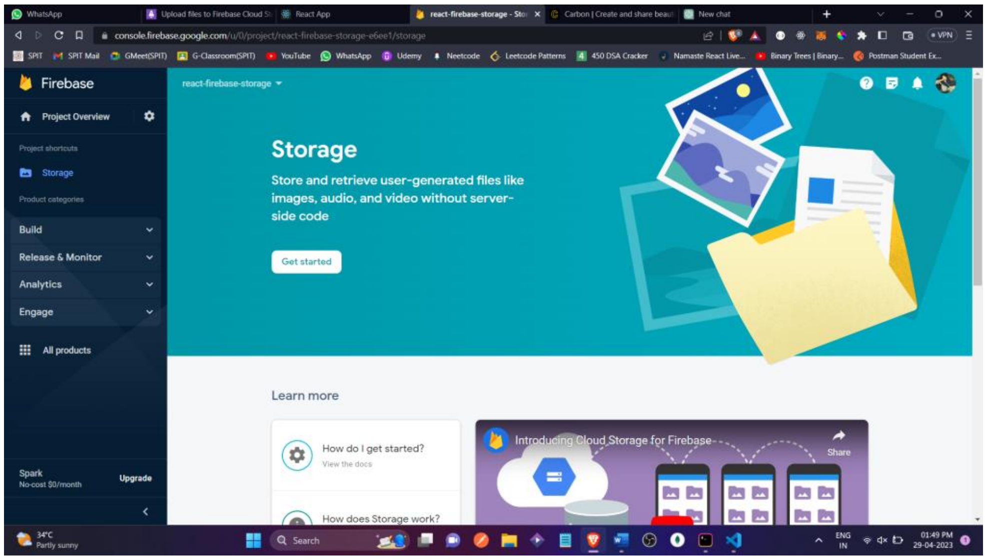
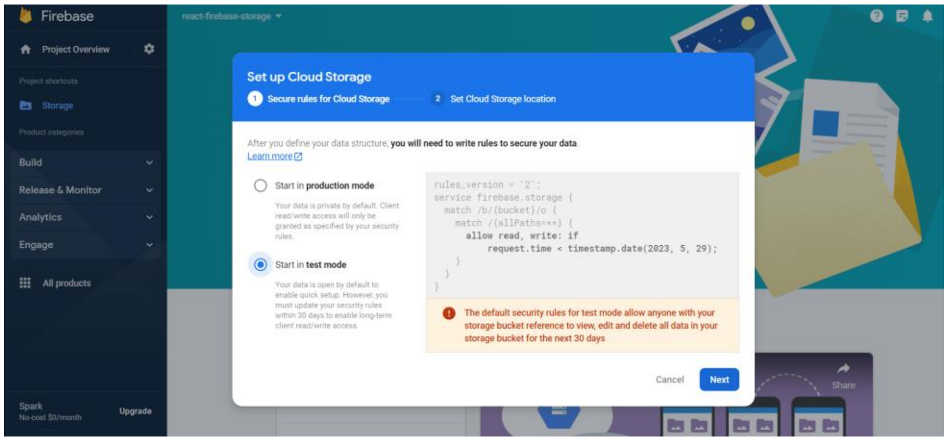


Create a Cloud Storage bucket
To use any of the Firebase services in your app, you must set them up for that project in Firebase Console. Therefore, Firebase knows that this app is using said product.

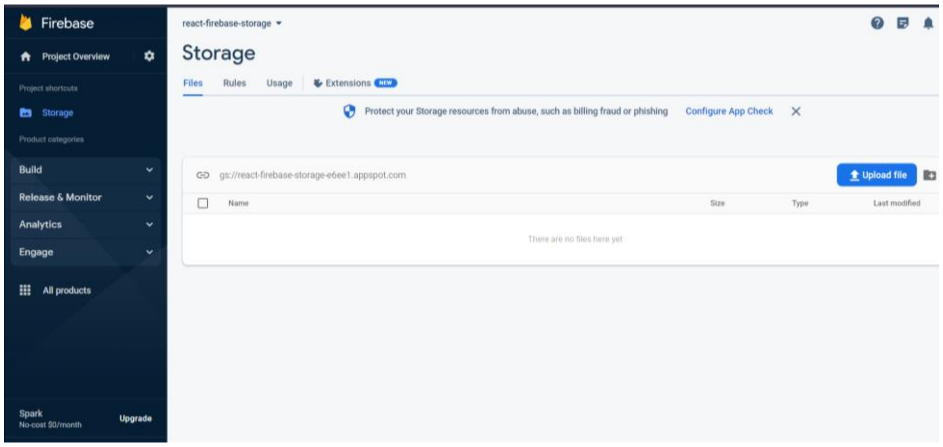
After copying the config code in Firebase console, click on **Go to console**. We will be shown an interface listing all the products we could use. On the left menu bar, click **Storage**:



Click on **Get Started**:

	
	 Firebase
	Project Overview
	Storage
	Store and retrieve user-generated files like images, audio, and video without server-side code
	Get started
	Learn more
	How do I get started?
	How does Storage work?
	Introducing Cloud Storage for Firebase
	
	For this purpose, we will choose **test mode**. But for production applications, you should choose **production mode** to limit who can read and write to the storage. **Click Next**:
	
	Set up Cloud Storage
	1 Secure rules for Cloud Storage — 2 Set Cloud Storage location
	After you define your data structure, **you will need to write rules to secure your data**. Learn more
	○ Start in **production mode**
	Your data is private by default. Client read/write access will only be granted as specified by your security rules.
	[✓ Start in **test mode**]
	Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.
	`rules_version = '2';`
	`service firebase.storage {`
	`  match /b/{bucket}/o {`
	`    match /{allPaths=**} {`
	`      allow read, write: if`
	`          request.time < timestamp.date(2023, 5, 29);`
	`    }`
	`  }`
	`}`
	The default security rules for test mode allow anyone with your storage bucket reference to view, edit and delete all data in your storage bucket for the next 30 days
	Cancel Next

	Select **Cloud Storage location** and click **Done**: Now, we can programmatically upload files to the Cloud Storage bucket and read those files:



3) Programmatically upload and read files

With that, everything is set up for us to write the code for uploading files. In App.js, we will start by importing the storage we exported from the Firebase config file, the methods we will use from firebase/storage, and the React useState Hook:

```
import { useState } from "react";
import { storage } from './firebase';
import { ref, getDownloadURL, uploadBytesResumable
} from "firebase/storage";
```

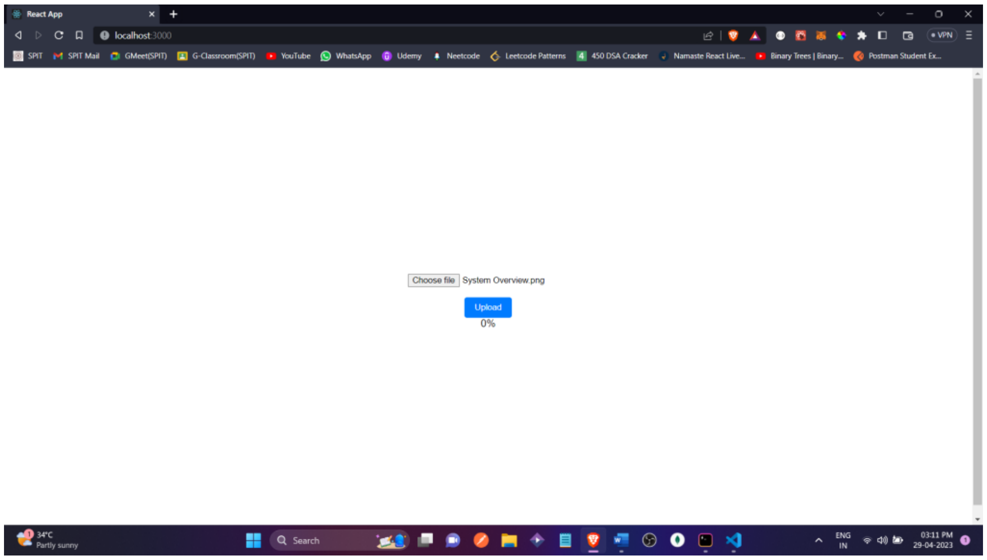
Let us write a function that will run when a user hits the **submit** button:

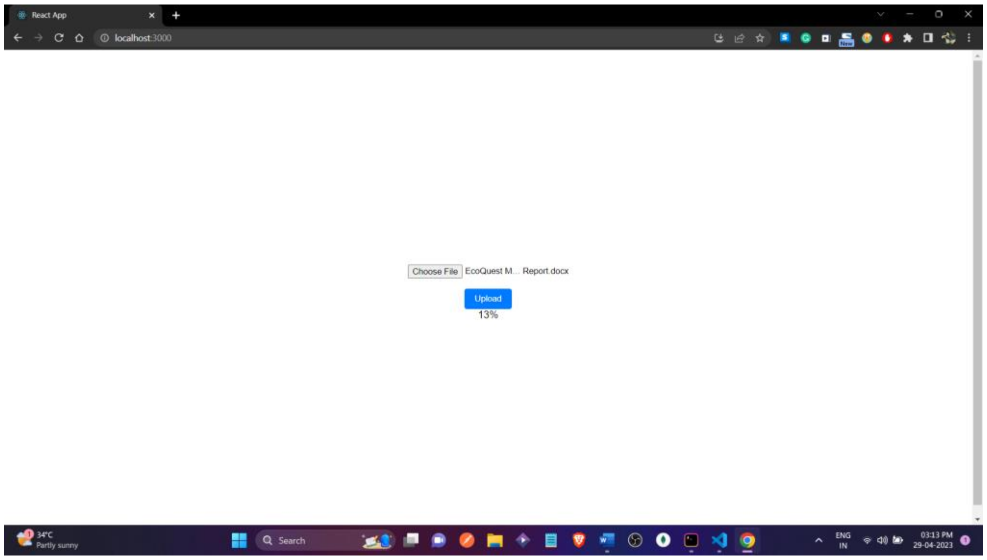
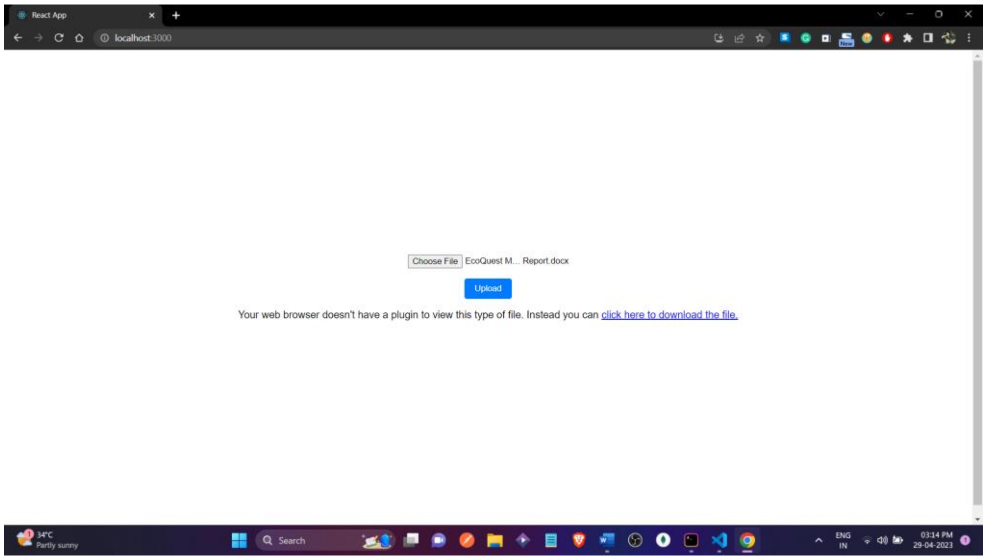
```
const [imgUrl, setImgUrl] = useState(null);
  const [progresspercent, setProgresspercent] =
useState(0);

  const handleSubmit = (e) => {
    e.preventDefault()
    const file = e.target[0]?.files[0]

    if (!file) return;

    const storageRef = ref(storage,
`files/${file.name}`);
    const uploadTask =
uploadBytesResumable(storageRef, file);
```

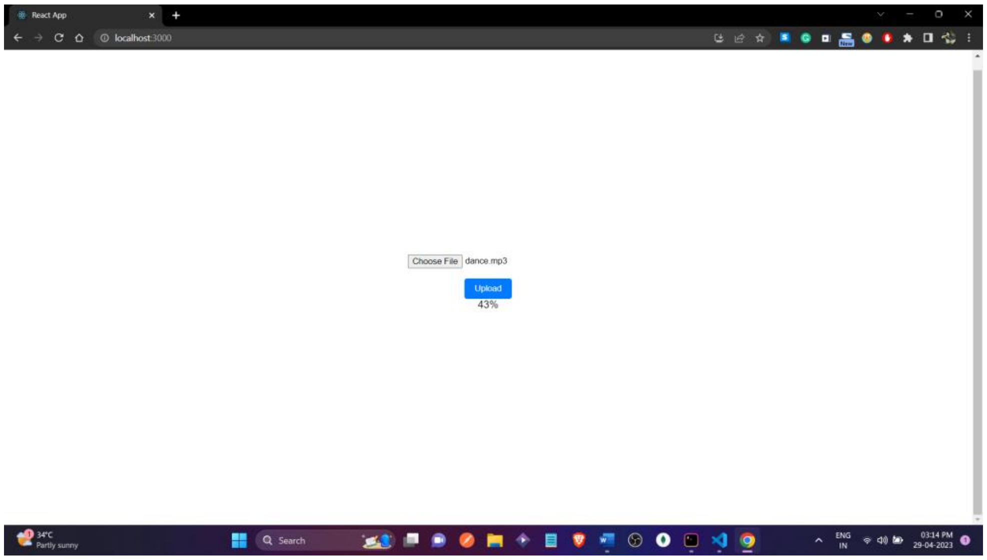
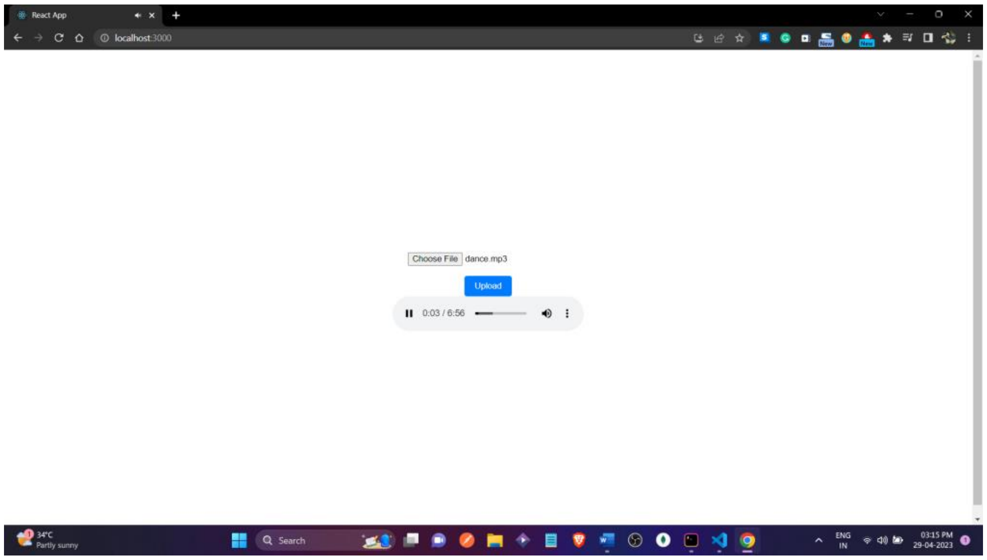
```
    uploadTask.on("state_changed",
      (snapshot) => {
        const progress =
          Math.round((snapshot.bytesTransferred /
snapshot.totalBytes) * 100);
        setProgresspercent(progress);
      },
      (error) => {
        alert(error);
      },
      () => {
        getDownloadURL(uploadTask.snapshot.ref).th
en((downloadURL) => {
          setImgUrl(downloadURL)
        });
      }
    );
  }
```

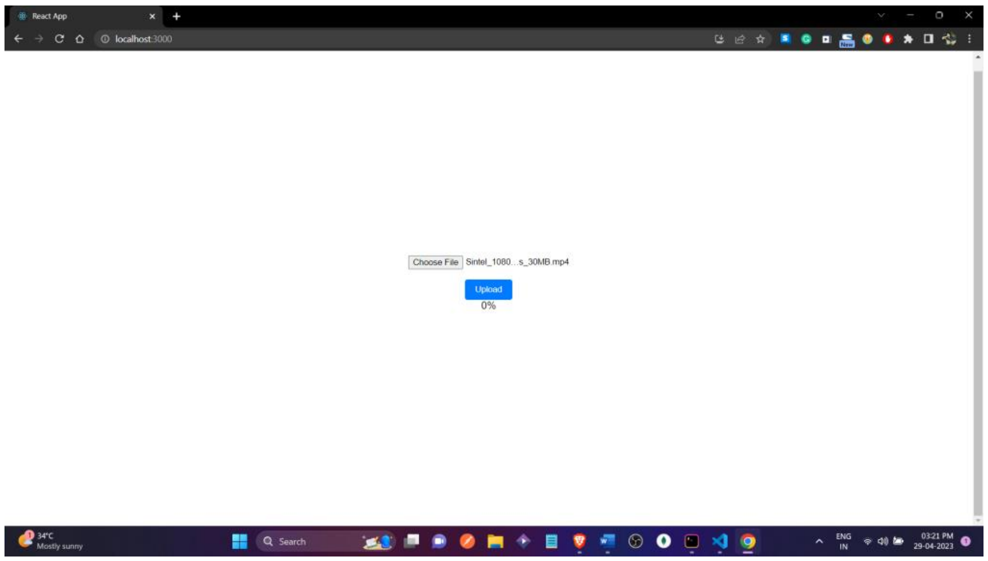
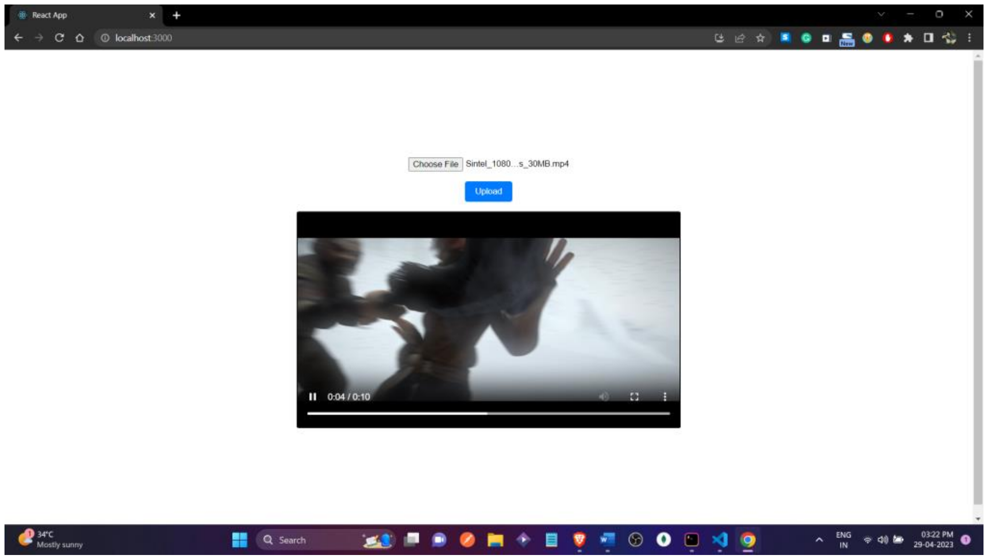
The full code for displaying the image and progress bar is shown below:

```
import './App.css';
import { useState } from "react";
import { storage } from './firebase';
import { ref, getDownloadURL, uploadBytesResumable
} from "firebase/storage";
import FileViewerComponent from
'./components/FileViewer';

function App() {
  const [mediaUrl, setMediaUrl] = useState(null);
  const [progresspercent, setProgresspercent] =
useState(0);

  const handleSubmit = (e) => {
    e.preventDefault()
    const file = e.target[0]?.files[0]
    if (!file) return;
    const storageRef = ref(storage,
`files/${file.name}`);
    const uploadTask =
uploadBytesResumable(storageRef, file);

    uploadTask.on("state_changed",
      (snapshot) => {
        const progress =
          Math.round((snapshot.bytesTransferred /
snapshot.totalBytes) * 100);
        setProgresspercent(progress);
      },
      (error) => {
        alert(error);
      },
      () => {
        getDownloadURL(uploadTask.snapshot.ref).th
en((downloadURL) => {
          setMediaUrl(downloadURL)
        });
      }
    );
  }

  return (
    <div className="App">
      <form onSubmit={handleSubmit}
className='form'>
        <input type='file' />
        <button type='submit'>Upload</button>
      </form>
      {
        !mediaUrl &&
        <div className='outerbar'>
          <div className='innerbar' style={{
width: `${progresspercent}%`
}}>{progresspercent}%</div>
        </div>
      }
      {
        mediaUrl &&
        <FileViewerComponent mediaUrl={mediaUrl}
/>
      }
    </div>
  );
}
export default App;
```

Output	Select any type of file, it will be uploaded to the bucket we created and will be visible once downloaded 

	 React App localhost:3000 Choose File EcoQuest M... Report.docx Upload 13% 34°C Partly sunny Search ENG IN 03:13 PM 29-04-2023  React App localhost:3000 Choose File EcoQuest M... Report.docx Upload 34°C Partly sunny Search ENG IN 03:14 PM 29-04-2023 Your web browser doesn't have a plugin to view this type of file. Instead you can click here to download the file.

	 Choose File dance.mp3 Upload 43%  Choose File dance.mp3 Upload 0:03 / 6:56

	 React App localhost:3000 Choose File Sintel_1080...s_30MB.mp4 Upload 0% 34°C Mostly sunny Search 03:21 PM 29-04-2023  React App localhost:3000 Choose File Sintel_1080...s_30MB.mp4 Upload 0:04 / 0:10 34°C Mostly sunny Search 03:22 PM 29-04-2023
Conclusion	This experiment tested the ability of databases to store binary files like media. Two methods were used - directly storing binary data in PostgreSQL, and using Firebase Storage and Firestore to upload files and generate download URLs. Both methods worked well, showcasing the flexibility of databases in storing binary files in different ways.
References	https://firebase.google.com/docs