|  |  |
| --- | --- |
| **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**    **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:    **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. |
| **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**:    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:    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:    Next, we will initialize a new React app and add Firebase to the project with the credentials provided:    **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      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:      In **App.js**, let us create a form for uploading files and a button for submitting:      **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**:    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**: |
|  | 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/${file.name}`);      const uploadTask = uploadBytesResumable(storageRef, file);      uploadTask.on("state\_changed",        (snapshot) => {          const progress =            Math.round((snapshot.bytesTransferred / snapshot.totalBytes) \* 100);          setProgresspercent(progress);        },        (error) => {          alert(error);        },        () => {          getDownloadURL(uploadTask.snapshot.ref).then((downloadURL) => {            setImgUrl(downloadURL)          });        }      );    }  The full code for displaying the image and progress bar is shown below:  import './App.css';  import { useState } from "react";  import { storage } from './firebase';  import { ref, getDownloadURL, uploadBytesResumable } from "firebase/storage";  import FileViewerComponent from './components/FileViewer';  function App() {    const [mediaUrl, setMediaUrl] = useState(null);    const [progresspercent, setProgresspercent] = useState(0);    const handleSubmit = (e) => {      e.preventDefault()      const file = e.target[0]?.files[0]      if (!file) return;      const storageRef = ref(storage, `files/${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).then((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 |
| **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> |