**EXPERIMENT NO. 06**

Shubham Golwal | 2020300015 | COMPS-D | ADBMS

**Aim**:

To develop a multimedia database

**Scenario**:

Create a database to perform CRUD operations and store any type of multimedia file and retrieve it back on demand. The database will act as organized storage of your data.

**Procedure**:

A) Using PostgreSQL

1. First create a python file which will be used as an application to interact with the database.
2. Make connections to the database.
3. Create a table with all required data related to the file such as its name, extension and one column of type bytea to store binary data.
4. If the user wants to store some data, ask for the path of the file, find out the file's name and extension, read the file,and insert it into the database using the insert command.
5. If the user wants to retrieve the data back, ask the user for filename and the extension of the file. Now retrieve the bytea data corresponding to provided data and write it to a binary file ending with provided extension.

B) Firebase Storage and Cloud Firestore

Step 1 : Create a Project on Firebase and install firebase cli on your system Step 2 : Create a react app and then cd into the directory and run the firebase init to initialize the firebase in your project

Step 3 : Select firestore and storage by using keyboard arrow buttons and space bar to select

Step 4 : Connect to existing firebase app created on the console

Step 5 : Complete the rules setup

Step 6 : Go to firebase console and create a database in firestore with test settings and create a bucket in storage as well

Now once the firebase is configured open the storage.rules file and change the access as follows

rules\_version = '2'; service firebase.storage { match /b/{bucket}/o { match /{allPaths=\*\*} {

allow read, write;

}

}

}

|  |
| --- |
|  |
| import React, { useState, useEffect } from 'react'; |
| import {app} from './base'; |

Change the firestore.rules file as follows

rules\_version = '2'; service cloud.firestore { match /databases/{database}/documents {

match /{document=\*\*} {

allow read, write;

}

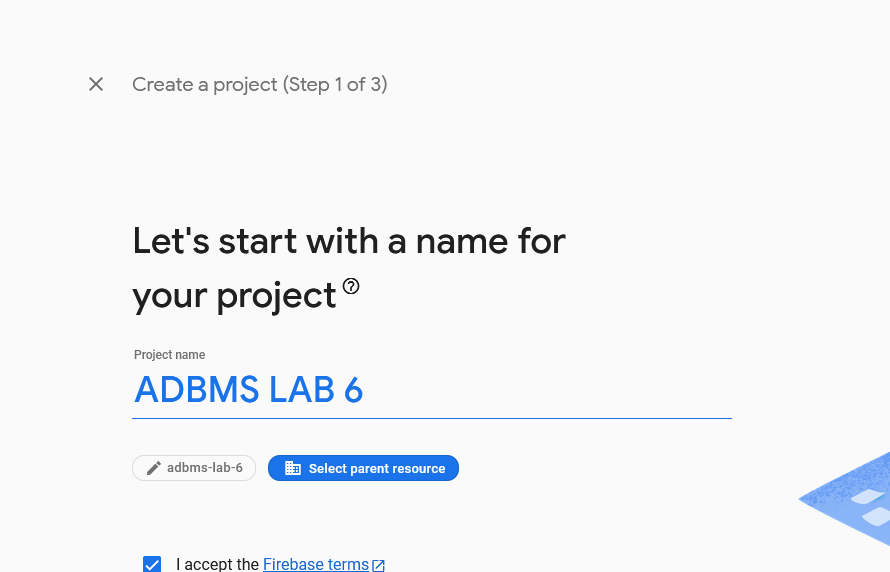
}

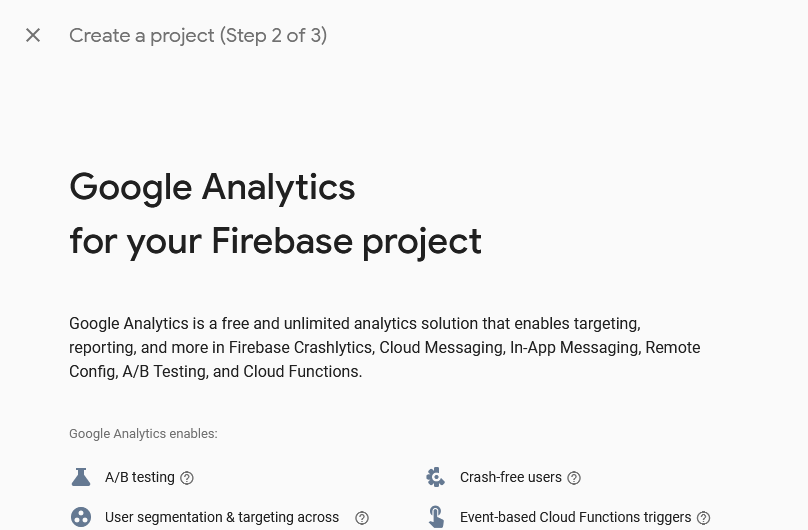
}

Now simply add your firebase configuration returned on the terminal into base.js file and import it in App.js

Now create a react component that takes image as input and username as input and it will store the image in firebase storage and the downloadable file for the image will be stored in the firestore database document along with the username provided by the user after submitting the data

INSTALLATION:





Graphical user interface, text, application, email

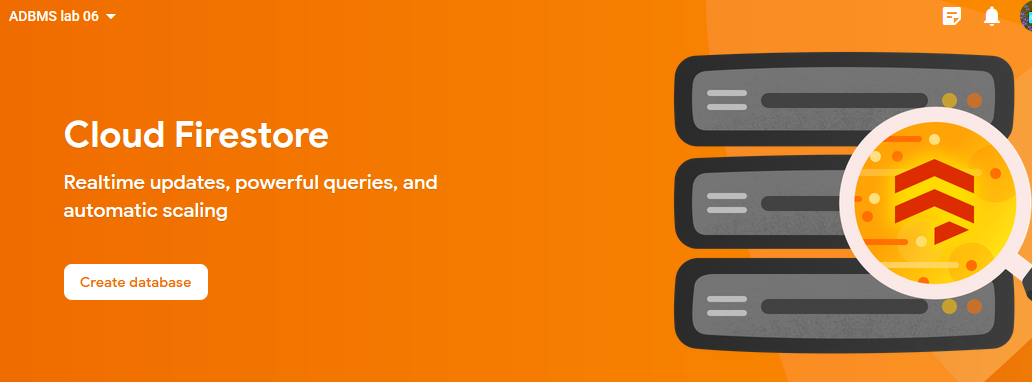
Description automatically generated

Application

Description automatically generated with low confidence

Graphical user interface, website

Description automatically generated



rules\_version = '2';

service cloud.firestore {

match /databases/{database}/documents {

match /{document=\*\*} {

allow read, write: if

request.time < timestamp.date(2023, 4, 27);

}

}

}

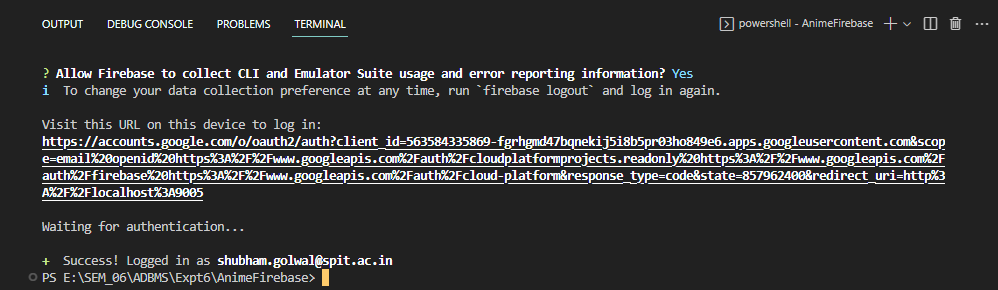


For testing purpose giving rights to everyone

Graphical user interface, text, application, chat or text message

Description automatically generated

For interaction with node



Server.js

const express = require("express");

const app = express();

const admin = require("firebase-admin");

const credentials = require("./key.json");

admin.initializeApp({

    credential: admin.credential.cert(credentials),

});

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

const db = admin.firestore();

*//POST request*

app.post("/create", async(req, res) => {

*try* {

        console.log(req.body);

        const id = req.body.email;

        const userJson = {

            email: req.body.email,

            firstName: req.body.firstName,

            lastName: req.body.lastName,

        };

*// const response = db.collection("users").doc(id).set(userJson);*

        const response = db.collection("users").add(userJson);

        res.send(response);

    } *catch* (error) {

        res.send(error);

    }

});

*//READ all request*

app.get("/read/all", async(req, res) => {

*try* {

        const usersRef = db.collection("users");

        const response = *await* usersRef.get();

        let responseArr = [];

        response.forEach((doc) => {

            responseArr.push(doc.data()); *// data->each doc*

        });

        res.send(responseArr);

    } *catch* (error) {

        res.send(error);

    }

});

*//READ one request*

app.get("/read/:id", async(req, res) => {

*try* {

        const usersRef = db.collection("users").doc(req.params.id);

        const response = *await* usersRef.get();

        res.send(response.data());

    } *catch* (error) {

        res.send(error);

    }

});

*//http://localhost:8080/create*

*//UPDATE request*

app.post("/update", async(req, res) => {

*try* {

        const id = req.body.id;

        const newFirstName = "updated first name";

        const userRef = db.collection("users").doc(id).update({

            firstName: newFirstName,

        });

*// const response = await usersRef.get();*

        res.send(response);

    } *catch* (error) {

        res.send(error);

    }

});

*//DELETE request*

app.delete("/delete/:id", async(req, res) => {

*try* {

        const id = req.params.id;

        const response = db.collection("users").doc(id).delete();

        res.send(response);

    } *catch* (error) {

        res.send(error);

    }

});

const PORT = process.env.PORT || 8080;

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}.`);

});

Key.json

{

    "type": "service\_account",

    "project\_id": "adbms-project-47fd8",

    "private\_key\_id": "a89762fa1160e1112cf34806d17a9d14f17d4dbd",

    "private\_key": "-----BEGIN PRIVATE KEY-----\nMIIEuwIBADANBgkqhkiG9w0BAQEFAASCBKUwggShAgEAAoIBAQCp0BNY6DR0X2Tn\nsSpyM/a91DXhhykLud8R6zvq3nrOPczH/c/5VZEZdWvZRw+eVOVn4xBCNN1J9wrm\nn3Snzis1o3DU/ITDq78kJQZFFJN25YoDIYgyfpzi1Lziv0Q0M1GmStAs7CkfWc+d\nke7GzR7HQirT9SuOWxl1E494YtQpMe579Lx9Iww4ergd2uWsVXwhVgcyUVrLzGJ5\nkaxgvyLeoLQHogtZQ1GyfowLtrxryykz2xsn+iKI2IMJ5SpXFMXV9XmJsqPB5w60\nVuKka/zCoCgnIcQ7gt3fkRE8KNDnVdKiI5wmw3A0OkhGYGFt94dCAQ/Zi07ZISoG\n+VzRODQlAgMBAAECggEAQLgaX5SkRUQEL5+gp0TxwbiKzokWVi9BetFWFTY0xLzd\n+szbwlHfyr2rXHSOWdBOXHE4RZtjSWjboTjdm4Tih/9iJfKV4QNohmyL/rEQYMyr\n9HSHtYL25SaiVKMD6gczE7dIrw5Uv+uZSpsPT3pmxBeOPhkfQBKtWacaU44LASW0\n9zNlHqOmMtSPMsGJyJV+rqAYPsWVI8onpZIENZ9X7k4sjO1KXnAn1JS9ZNe7pTqu\n/RSNEb2zWNjWCqqc2NS5XNxDsrgowB8O7/RNhplDE74iXwiOT9rSfEGdINADNk35\nnScP4fp/cY450ZZoUnnkbaIo3vtznw00xy0uZeIDowKBgQDbOSNgdl6PjmI7rjeZ\nkYyauBTGWgvuPFBNmImE7vLGRYXOsklUz5lHnhnIOrC6uCTRRBtmKhiAwpJOV+mI\nf5UtUALjnfSh4NXCYNKvBGHuG3xha6JR/cKtmY6i4iwnw0U5SvnUKbdqPBSkEoox\n4KOAlZ0XlLcSCJBKzxvCBAoB4wKBgQDGTO9qy5NUBsbYeIwSd4ROK2iCWAY/rjGH\ng6jlGZE6b5WZ3LhMpUKPedw2+db2e/+M7rvYvO80xLWYdg8sFXrZMem0Pk0aKW3W\nebPbCsgxpMkPKsY/3gBIopRYZ087u3bnBz46YTAwQc/FAx1aJ3bQGzRHtYqlphw6\nLmlBNl0wVwKBgGw/nZlfObg3PYhssWWyOjSX2yJKioOb4qc+QAGVH+oJoFX9MTqg\nu/ZiOibl3CiHDx8KO2FKPgkuiqIGxPkyzo2uaY7Ni2N3qvWta7siNd0mU1ejl2ZX\nII8qOQVzGwyTXpY40H4UD0501lhiEz0nyHSQ7B4dv/w2oFFgTTuTVQGzAoGBAJ0f\naLTeCSaTGAt9uL8JJDQsAOsGhC0BAQ23LKQa1g/PSmJXmVBhwZOdoveMjt9O8PXi\nU8UwLvlIw/rB0NPFUUm3Ws4Ob1lDalx+ohv1NxreztDNu7fvV5o1+AT0FrvllhkI\njC4FxAD4pIAeL6u3iVTruUj4d5s5OzR+gbfCM6SRAn9ggUyl3r+1NA633qE4ESxj\nl/2sE5PtVJe0pygzCldrod2gOVDhmVfRuCpERgpfH7fWnOZIhUMOtk7uOyFn/bFk\nrsXj9m9YD8sFi4OrpZdGWGRKcfNCfY856WALZeOujml+ofT/iI1a5UjcXEg7Kgn2\n+59ZRhx/vlsmQ89giTDs\n-----END PRIVATE KEY-----\n",

    "client\_email": "firebase-adminsdk-lk283@adbms-project-47fd8.iam.gserviceaccount.com",

    "client\_id": "107264716051288400733",

    "auth\_uri": "https://accounts.google.com/o/oauth2/auth",

    "token\_uri": "https://oauth2.googleapis.com/token",

    "auth\_provider\_x509\_cert\_url": "https://www.googleapis.com/oauth2/v1/certs",

    "client\_x509\_cert\_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-lk283%40adbms-project-47fd8.iam.gserviceaccount.com"

}

Text

Description automatically generated

OUTPUT :

Performed CRUD operations:

A screenshot of a computer

Description automatically generated with medium confidence

Server is started:

Text

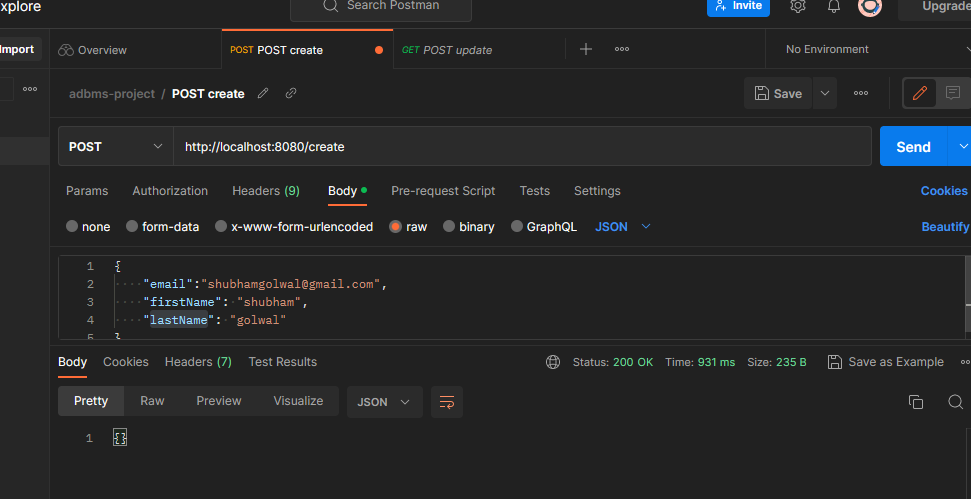
Description automatically generated

POST:

A screenshot of a computer

Description automatically generated with medium confidence

Request sent from postman



Server received the request

Text

Description automatically generated

Added on firebase

Graphical user interface, text, application

Description automatically generated

Read:

A screenshot of a computer

Description automatically generated with medium confidence

Request from postman

A screenshot of a computer

Description automatically generated with medium confidence

UPDATE:

Request from postman

Graphical user interface, text

Description automatically generated

Updated on database

Graphical user interface, text, application

Description automatically generated

DELETE:

request

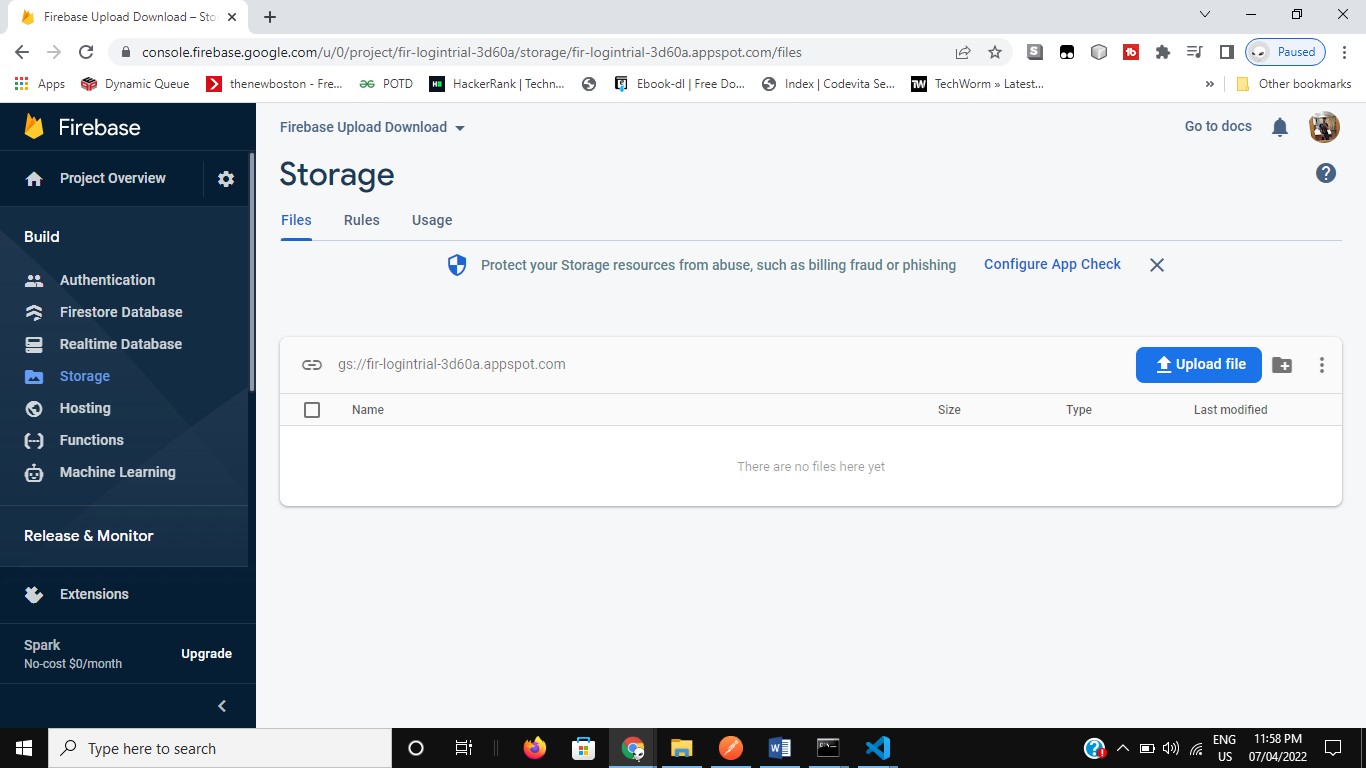
Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

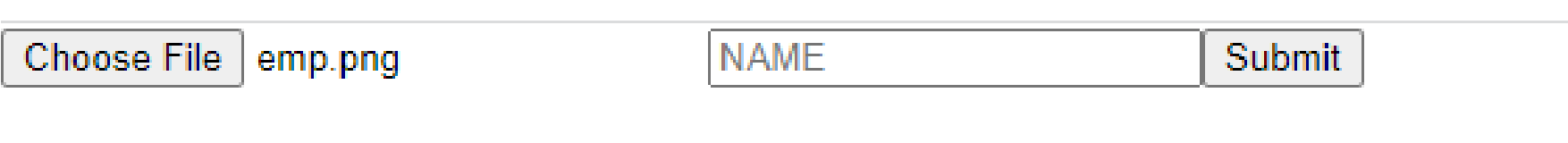
Firebase storage empty i.e. nothing in the bucket



Firestore collection empty

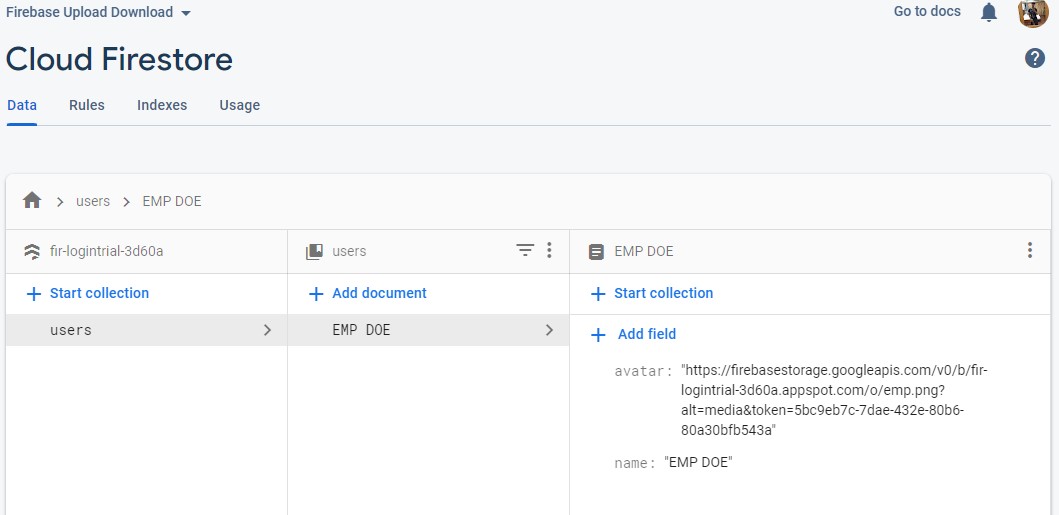
Choose an image file : as soon as file is chosen, it will be stored in the storage bucket

File Chosen emp.png



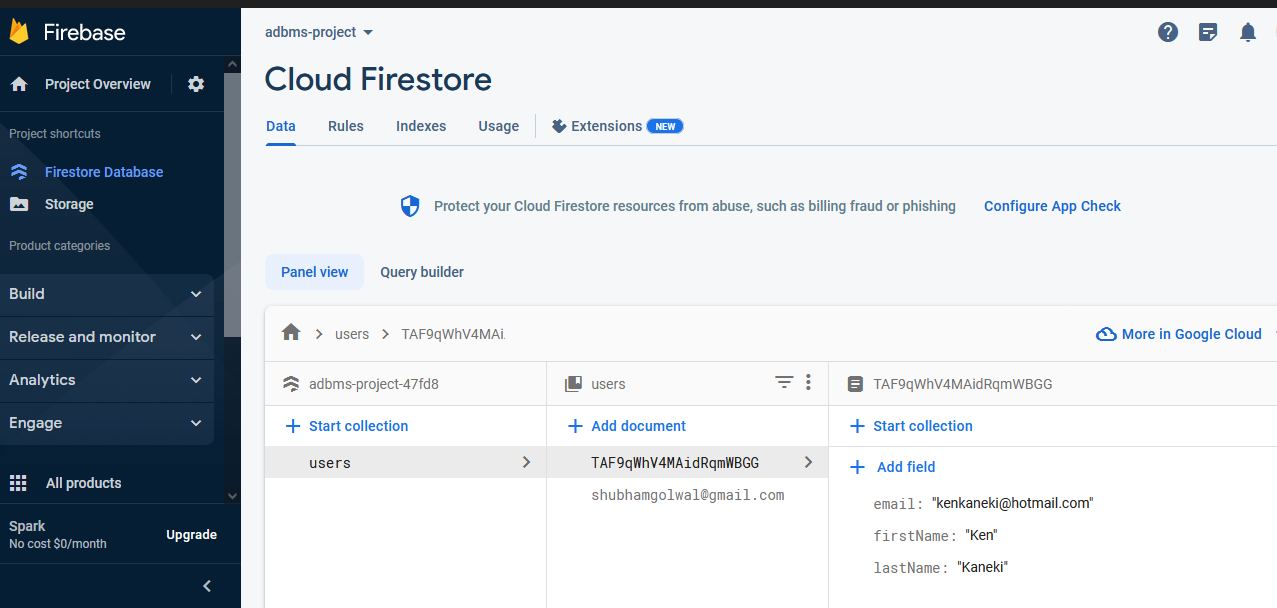
New file named emp.png added to the storage bucket

Collection users created with EMP DOE document and the document contains name and the downloadable url of the image uploaded

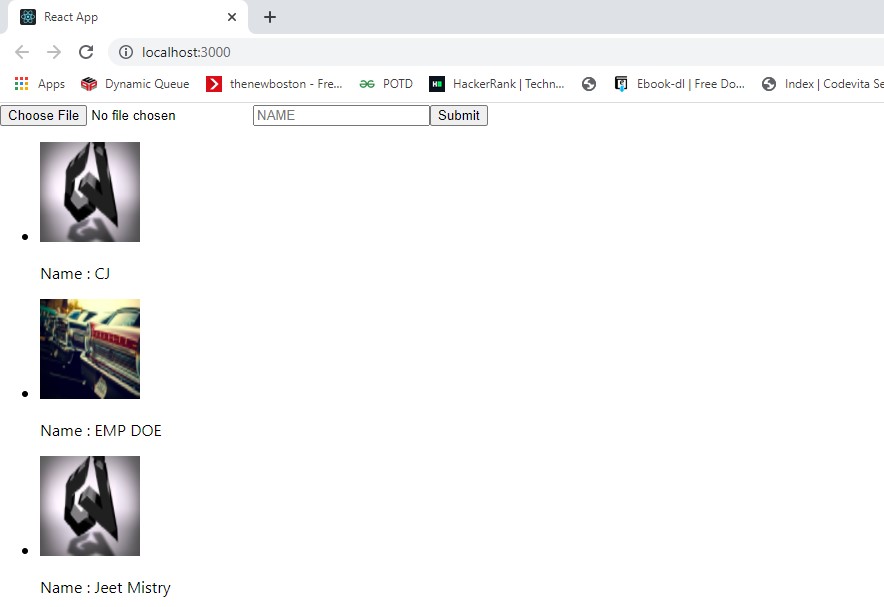


Now the uploaded image and user will be listed in the homepage of the app once successfully uploaded

Submitting multiple images to see multiple images in the app



App



**Conclusion**: From this experiment, we

* Learnt that Databases can store binary files like media files.
* Used Two methods were used: PostgreSQL and Firebase.
* PostgreSQL stores entire binary data in the database, increasing its size but not requiring external storage.
* Firebase uses two components: Firebase Storage to upload images and generate downloadable URLs, and Firebase Firestore to store data in document and key-value form and retrieve images in real-time.