Experiment 6

Ojas Patil
2019130048
TE COMPS

Aim: To develop a multimedia database
Scenario: Create a database to store any type of multimedia file and retrieve it back on demand. Thai database will act as organised 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.

Code:

```
import psycopg2

dbConn = psycopg2.connect(database="", user="", password="",
host="", port=5432)
dbConn.autocommit = True
dbCursor = dbConn.cursor()

try:
    dbCursor.execute("""
    CREATE TABLE mediafiles (
        name text,
```

```
        extension text,
        media bytea
    );
    """)
except:
    pass

while True:
    choice = int(input("""Enter 1 to insert mediafile and 2 to
retrieve mediafile: """))
    if choice  == 1:
        path = input("Enter path to mediafile: ")
        name, extension = path.split('\\')[-1].split('.')
        extension = '.' + extension
        img = open(path, 'rb')

        dbCursor.execute(f"""INSERT INTO mediafiles (name,
extension, media) VALUES( %s, %s, %s)""", (name, extension,
psycopg2.Binary(img.read())))
        dbConn.commit()

        img.close()

    elif choice == 2:
        name = input("Enter name of file to be retrieved: ")
        extension = input("Enter its extension: ")
        dbCursor.execute(f"SELECT extension, media FROM mediafiles
where name='{name}' AND extension='{extension}';")
        blob = dbCursor.fetchall()[0]
        open(f'Retrieved{name}{blob[0]}', 'wb').write(blob[1])

    else:
        print("Exitting...")
        exit(0)
```

Output:

A. Terminal

```
C:\Users\ojasp\Documents\Sem VI\ADBMS>python -u "c:\Users\ojasp\Documents\Sem VI\ADBMS\Exp6.py"
Enter 1 to insert mediafile and 2 to retrieve mediafile: 1
Enter path to mediafile: C:\Users\ojasp\Documents\Sem VI\ADBMS\Image.jpg
Enter 1 to insert mediafile and 2 to retrieve mediafile: 2
Enter name of file to be retrieved: Image
Enter its extension: .jpg
Enter 1 to insert mediafile and 2 to retrieve mediafile: 0
Exitting...
```

B. Database

Data Output | Explain | Messages | Notifications

	name text	extension text	media bytea
1	Image	.jpg	[binary data]

<u>B) Firebase Storage and Cloud Firestore</u>

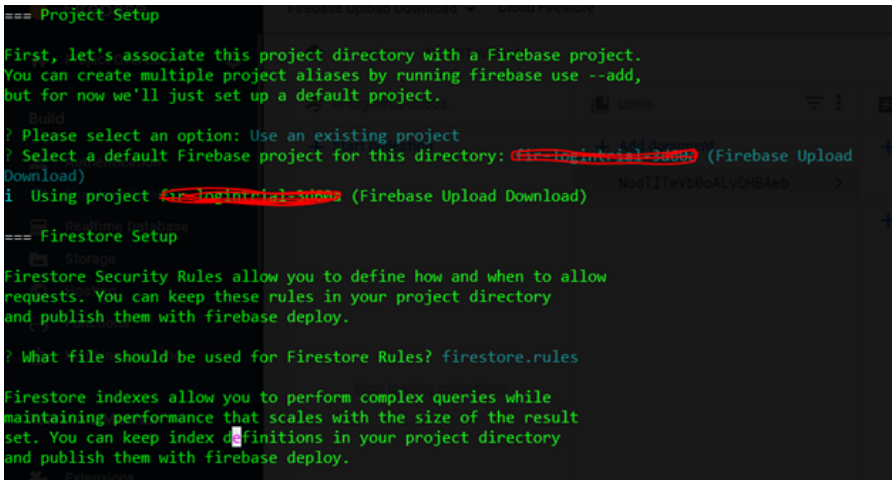
Step 1 : Create a Project on Firebase and install firebase cli on your system

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;
    }
  }
}
```

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

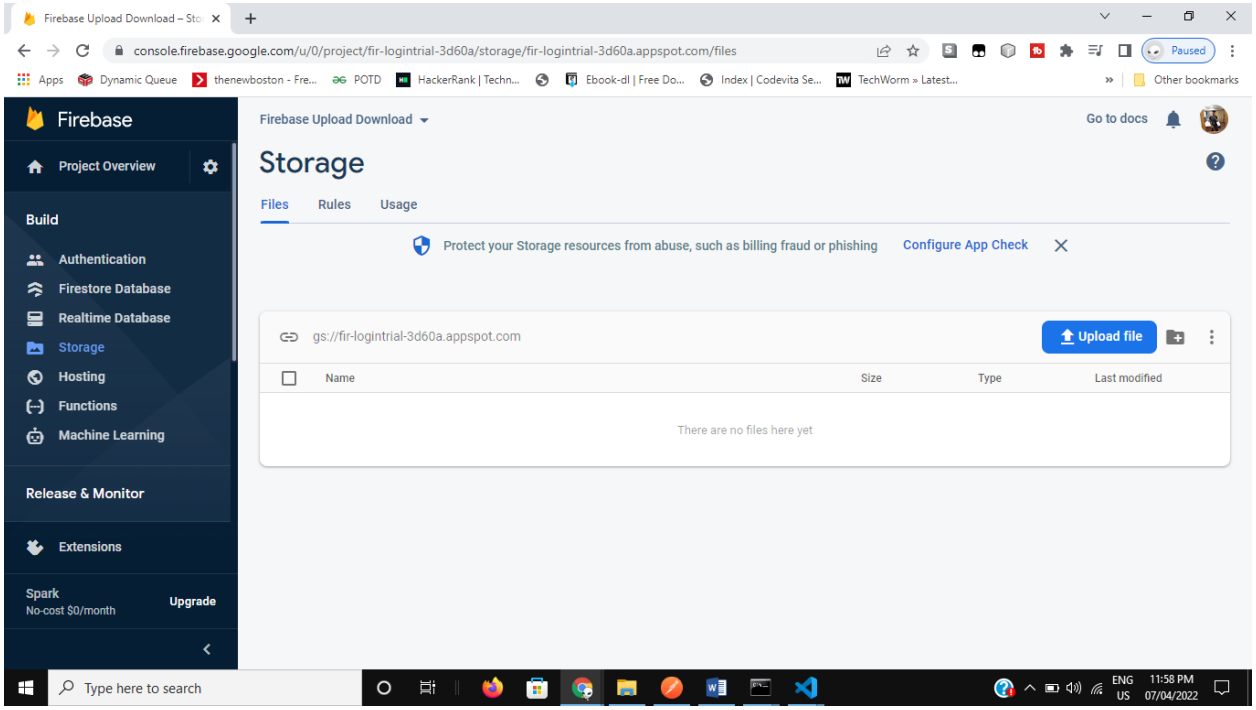
```
import React, { useState, useEffect } from 'react';
import {app} from './base';
```

```
import 'firebase/compat/firestore';

const db = app.firestore();

function App() {

  const [fileUrl,setFileUrl] = useState("");
  const [users,setUsers] = useState([]);

  const onFileChange = async(e)=>{
    const file = e.target.files[0];
    const storageRef = app.storage().ref()
    const fileRef  = storageRef.child(file.name);
    await fileRef.put(file);
    const f = await fileRef.getDownloadURL()
    console.log(f);
    setFileUrl(f);

  }

  const onSubmit = (e)=>{
    e.preventDefault();
    const username = e.target.username.value;
    if(!username){
      return;
    }
    db.collection("users").doc(username).set({
      name:username,
      avatar:fileUrl
    })
  }

  useEffect(() => {
    const fetchUsers = async()=>{
      const usersCollection = await db.collection("users").get()
      setUsers(usersCollection.docs.map(doc=>{
        return doc.data()
      }))
    }
```

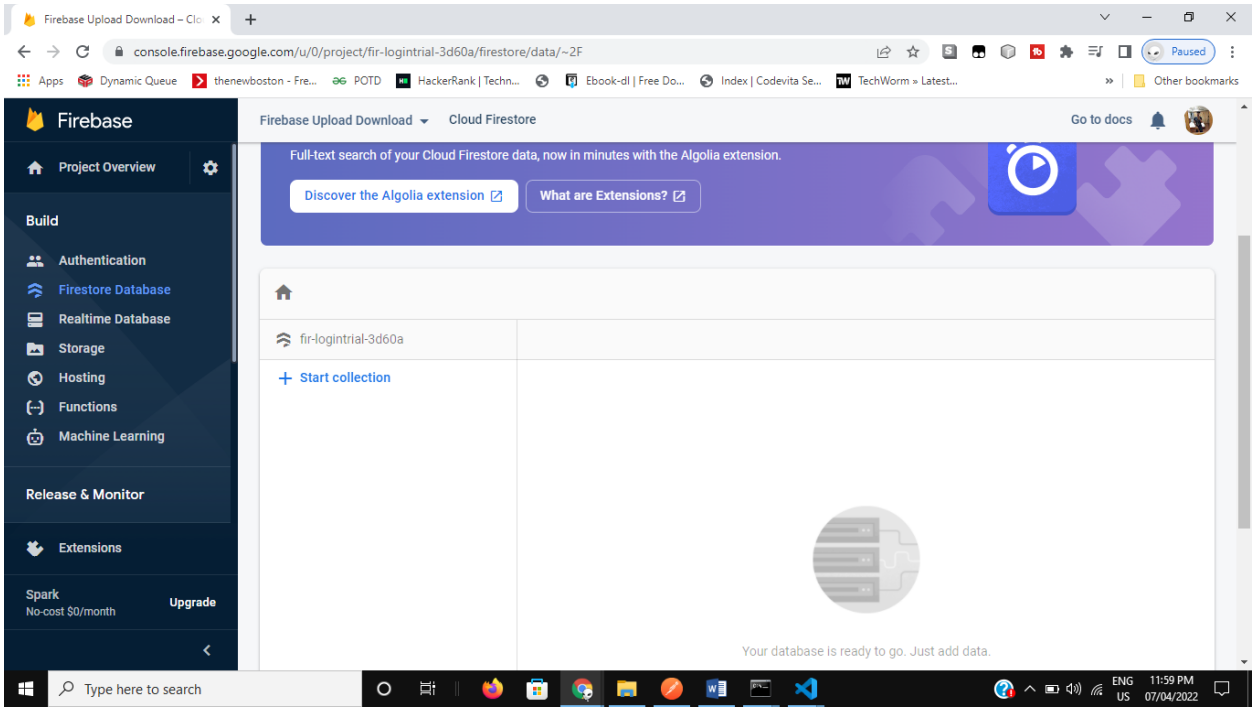
```
    fetchUsers();
  }, [])

  return (
    <>
      <form onSubmit={onSubmit}>
        <input type="file" onChange={onFileChange}/>
        <input type="text" name="username" placeholder="NAME" />
        <button>Submit</button>
      </form>

      <ul>
        {users.map(user=>{
          return(
          <li key={user.name}>
            <img width="100" height="100" src={user.avatar}
alt={user.name}/>
            <p>Name : {user.name}</p>
            </li>
          )
        })}
      </ul>
    </>
  );
}

export default App;
```

OUTPUT :

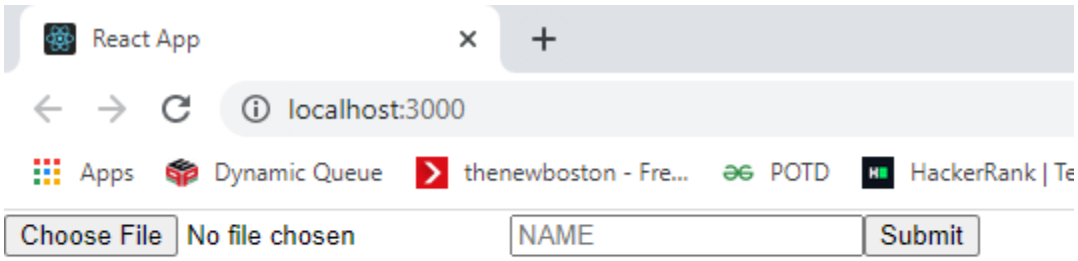
Firebase storage empty i.e. nothing in the bucket



Firestore collection empty

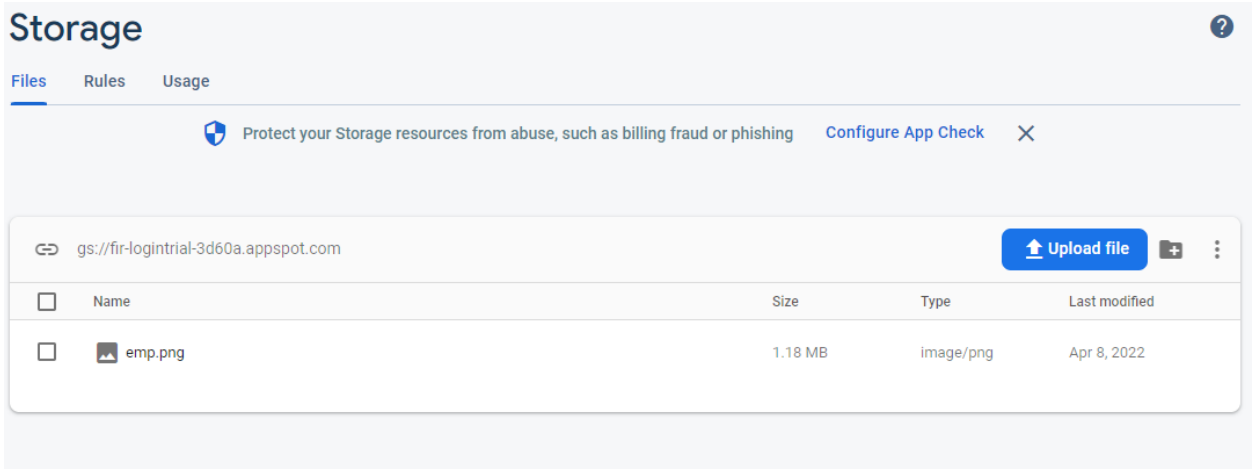


React App

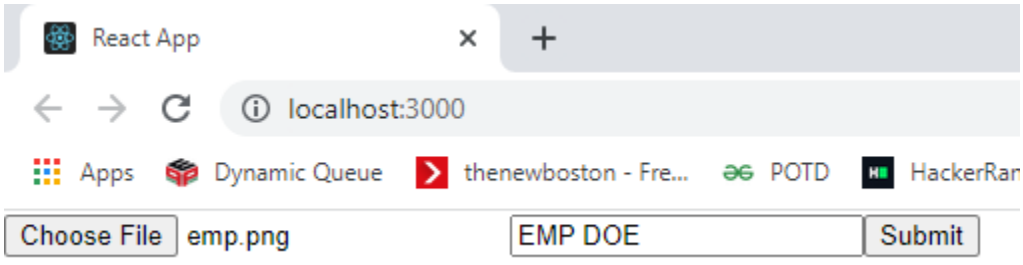


Choose an image file : as soon as file is chosen, it will be stored in the storage bucket
File Chosen emp.png

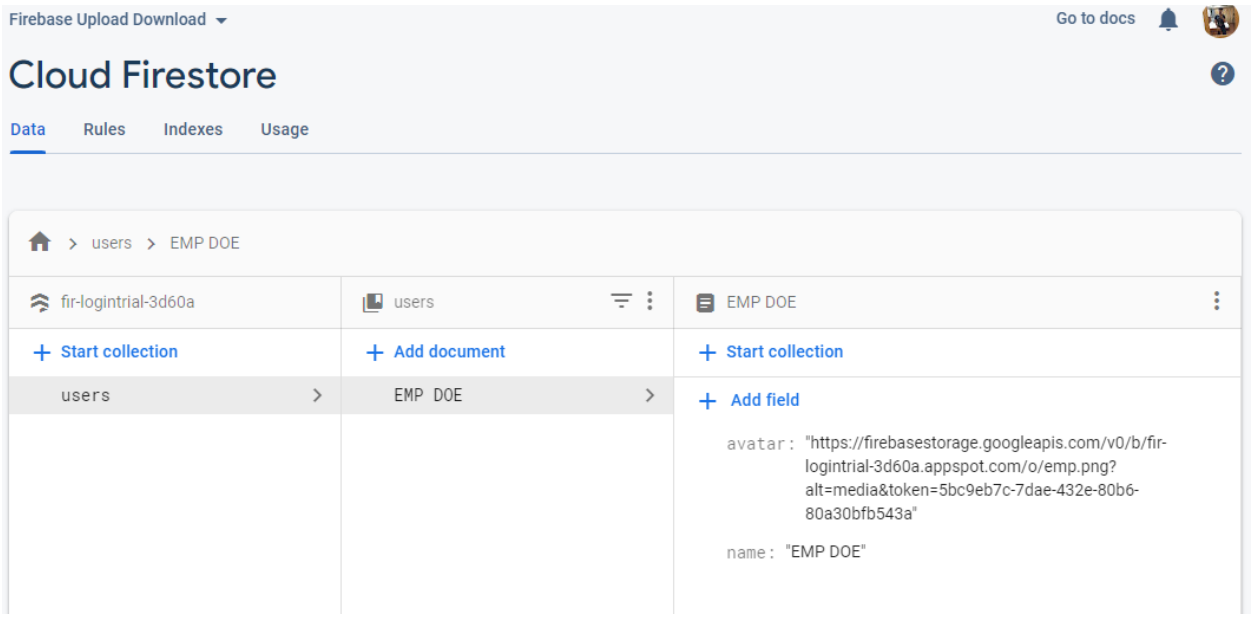
New file named emp.png added to the storage bucket



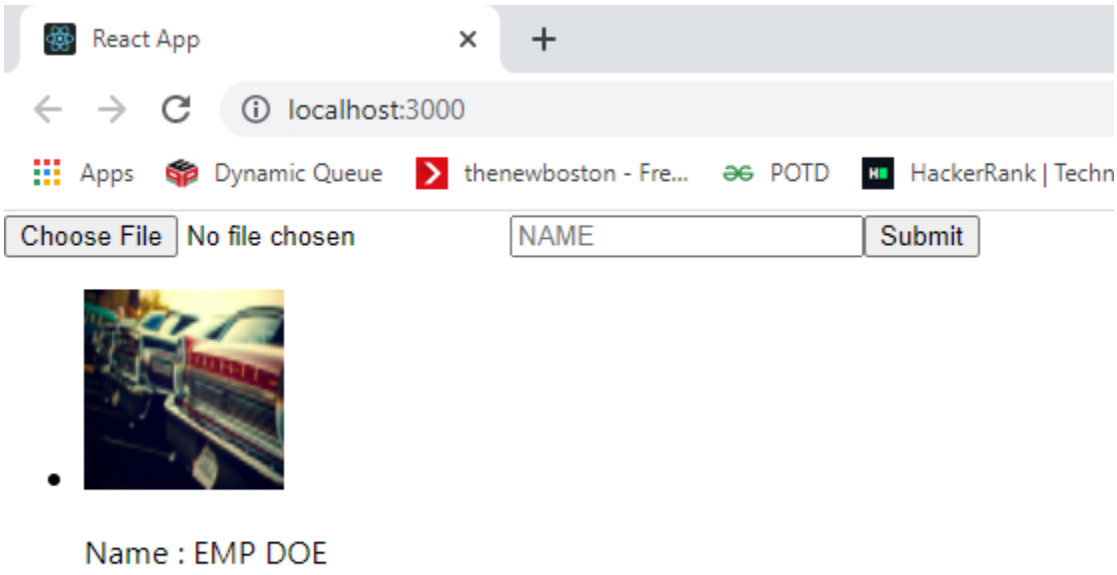
Now submitting the file with adding username as EMP DOE



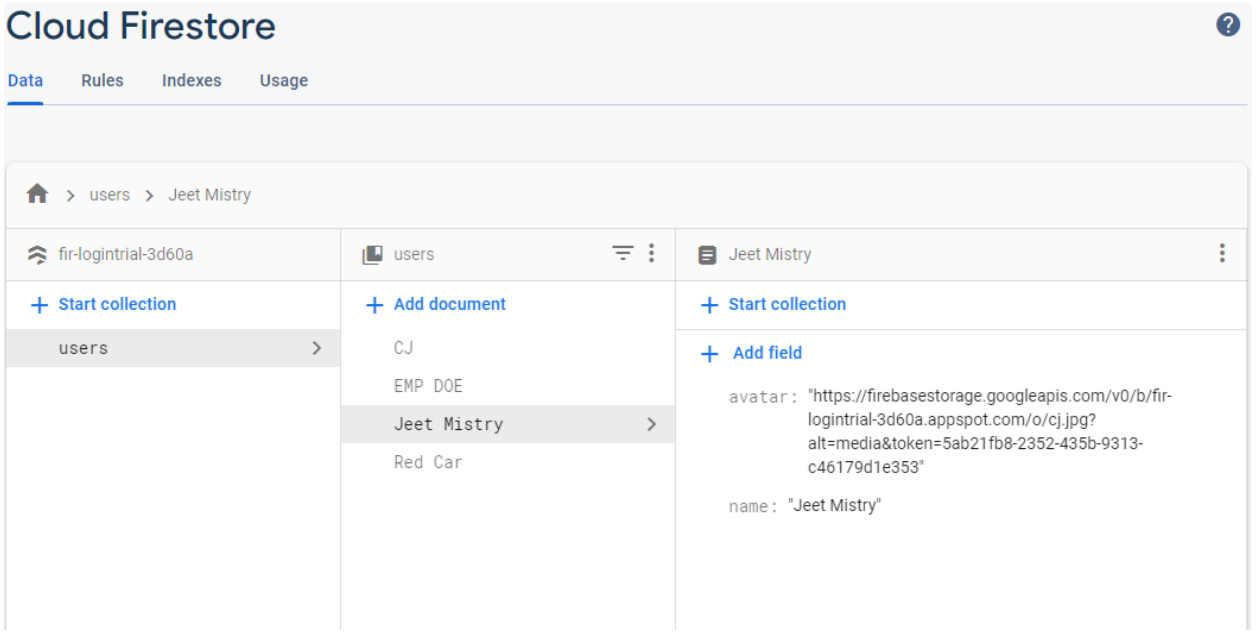
Collection users created with EMP DOE document and the document contains name and the downloadable url of the image uploaded



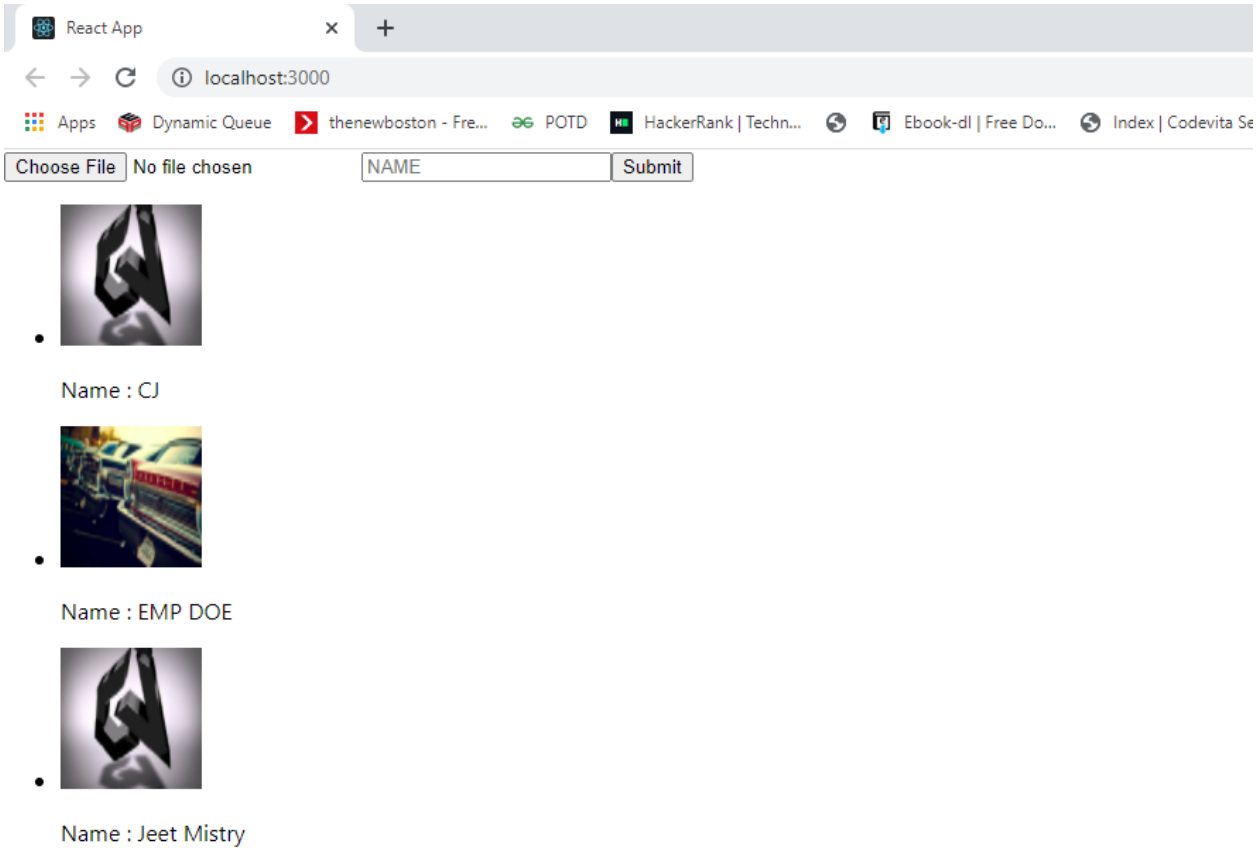
Now the uploaded image and user will be listed in the homepage of the app once successfully uploaded



Submitting multiple images to see multiple images in the app



App



Conclusion:

From this experiment, we unlocked the potential of databases to store binary files as well. We learnt to store media files in the database. In the two methods of postgreSQl and FireBase, we used two methods to achieve this functionality, one is to put entire binary data into the database like we did in PostgreSQL, thus increasing its size but doesn't require any external storage system. In the case of FireBase, we used two cloud components of firebase, the first one is the firebase storage in which we uploaded our images from the application and then we generated the downloadable url of the images to store it in the second component which is the firebase firestore which stored the data in document and key value form and the retrieved the images from the database in real time.