SECURE FILE STORAGE ON CLOUD USING HYBRID CRYPTOGRAPHY

PROJECT REPORT

Submitted by

MANJULA R. - (2019272022)

in partial fulfilment for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY, CEG

ANNA UNIVERSITY, CHENNAI 600 025

MAY, 2022

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Report titled "**SECURE FILE STORAGE ON CLOUD USING HYBRID CRYPTOGRAPHY**" is the bonafide work **MANJULA R. (2019272022)** who carried out the work under my supervision. Certified further that to best of my knowledge the work reported herein does not from part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

PLACE : CHENNAI

Date : 05 – 05 – 2022

Dr. L. SAI RAMESH

TEACHING FELLOW

PROJECT GUIDE

DEPARTMENT OF IST, CEG

ANNA UNIVERSITY

CHENNAI – 600 025

COUNTER SIGNED

Dr. S. SRIDHAR

HEAD OF THE DEPARTMENT

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING, GUINDY - ANNA UNIVERSITY

CHENNAI – 600 025

TABLE OF CONTENTS

ABSTRACT

Cloud computing aims to provide computing resources, massive data storage capacity and flexible data sharing services. Cloud is used in various fields like industry, military, colleges, etc. for various services and storage of huge amount of data. Data stored in this cloud can be accessed or retrieved on the users request without direct access to the server computer. But the major concern regarding storage of data online that is on the cloud is the Security. This Security concern can be solved using various ways, the most commonly used techniques are cryptography. But sometimes a single technique or algorithm alone cannot provide high-level security. So, we are using a combination of multiple cryptographic algorithms. In this proposed system AES (Advanced Encryption Standard) and RSA (Rivest Shamir Adleman) algorithms are used to provide security to data. File during encryption is split into parts. These individual parts of the file will be encrypted using different encryption algorithm simultaneously with the help of multithreading technique. Our methodology gives the better security and protection of customer data by storing encrypted data on a single cloud server using AES and RSA algorithm.

CHAPTER I

INTRODUCTION

1.1 General

Cloud computing is emerging as the most suitable for individuals and organizations to access inexpensive, scalable, and on-demand computing resources, applications and data storage services. Cloud storage systems such as Dropbox, Google Drive, Microsoft OneDrive, etc., enable users to remotely store a large volume of data that can be accessed and shared among users, regardless of time and location constraints.

Cloud storage is essentially one of the simplest applications of cloud technology available. As mentioned above all of these services are cloud storage devices. They do one thing that is to allow you to store data on the cloud. The cloud storage is to put it simply someone else's computers. If you have a terabyte of data you need to buy a suitably large hard drive to store it locally. If the hard drive fails the data get erased, so we need some kind of backup system which means more hard drives. This can get expensive, particularly in terms of hardware and the man-hours necessary to test and maintain those backups.

Cloud computing using cloud computers to perform computational tasks. This is basically anything other than storage, including simple applications and high-end processing-intensive tasks. For example, simple Software As A Service (SaaS) applications are ran on cloud platform. Using webmail such as Gmail or Outlook.com is a form of cloud computing, as using Office 365 for office apps that are hosted online. At the other end of the scale, you have cloud-based processing tasks, where you can rent time using their processing power to complete your tasks. For example, some research facilities use this to complete complex calculations, and some video production companies use this to render videos faster than they could using local hardware.

1.2 Problem Statement

With growing popularity of cloud computing the number of enterprises and individuals shifting toward the use of cloud has increased rapidly. As a result, a vast amount of important personal information and critical organization data, such as personal health records, government documents, and company finance data, etc., are transmitted across the Internet and stored in cloud servers.

However, outsourcing sensitive data suffers from critical security threats, privacy, and access control problems. These are common concerns of using cloud services. When data owners migrate their sensitive data to the cloud, they lose an element of control over their data. Users have no guarantee about the way these sensitive data will be treated and protected by cloud providers.

Although the cloud provides users with the convenience of data access across multiple devices, by using cloud services user data are vulnerable to variety of malicious attacks and threats. Security incidents occurs frequently. One feasible solution to overcome these problems is to use cryptography.

1.3 Objective

All sensitive data have to be encrypted by data owners prior to storing them into the potentially untrustworthy cloud. The strength of the encryption scheme is largely dependent on the strength of the key management technique used. The security of the encryption scheme lies on the secrecy of the keys that are known only to the users authorized to read their respective data and not only on the secrecy of the encryption algorithm used.

Given the amount of data being stored and shared in cloud and the increasing number of data users, designing a cryptographic scheme for cloud storage that meets the requirements of security, efficiency and flexibility, which is consider to be a challenging task.

CHAPTER II

LITERATURE REVIEW

A literature survey was done by analysing various research papers to understand the limitations and workings from those papers will help to create a better system.

2.1 Cloud Storage

Storing the documents or files on the cloud is the primary goal of every research works. Those cloud storage will allow the files to be stored on any cloud platform which can be easily accessible by the users. There will be data owner who is also one of the users on the system, where, they store the documents from the remote computer into the cloud service which provides the storage for the files.

2.2 Encryption and Decryption

Encryption techniques that followed by Mr. Uttam Kumar & Mr. Jay Prakash, in File Storage on Cloud, is the symmetric key cryptography which provides a simple encryption algorithm that is developed using the AES and DES encryption techniques. Decryption is the same reverse process of those encryption technique which uses the same algorithms.

This research carried out by Osama Ahmed Khashan in Secure Outsourcing and Sharing of Cloud data, encryption algorithm used in the system is OutFS file system where it is designed on the top of third-party cloud tools to allow data owner outsourcing their files in an encrypted manner. This performs mostly on the kernel file system. This builds on a standard hierarchical view of files and directories, and interacts easily with standard system calls, like read, write, open and save. OutFS is

also designed to perform the integrity and encryption operations on a per-file basis using a per-file key to allow a user with ability to select the appropriate file to access or share.

2.3 File Sharing

According to Mr. Uttam Kumar & Mr. Jay Prakash, in File Storage on Cloud, AES and DES algorithm for encryption and decryption, the sharing of the keys is also made public, as it uses the symmetric cryptographic techniques which may help the other users to share, view and download the files on the system. By this way the files on the storage is shared with other users other than data owner of the file.

This research carried out by Osama Ahmed Khashan in Secure Outsourcing and Sharing of Cloud data, here the data owner must guarantee that all operations of a file sharing are processed in a secure and authenticated manner. In addition, it can only be accessed by legitimate users who own valid keys to decrypt the files. Moreover, the secret keys are not stored or known by the cloud storage provider or other users. In the sharing encryption scheme, data files are encrypted using the AES symmetric block cipher.

CHAPTER III

SYSTEM DESIGN AND ARCHITECTURE

3.1 Architecture Diagram

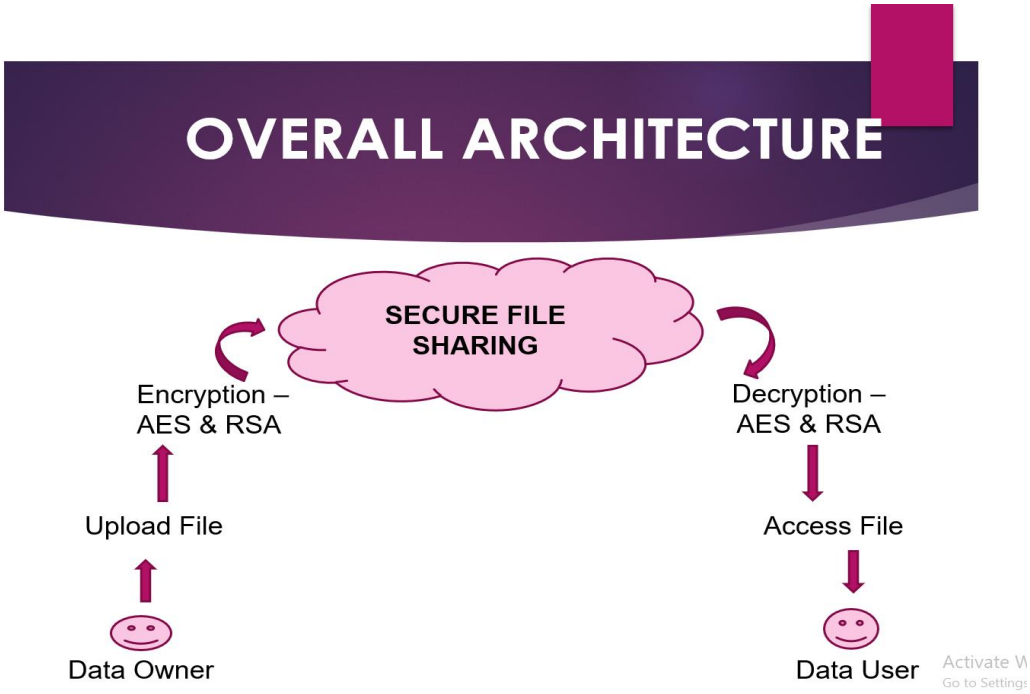

Figure (3.1) Overall System Architecture

3.2 Architecture Explanation

Figure (3.1) depicts the entire workflow of the system and the process which is taken place in the system. First, the user who may be the data owner tries to upload the file on the storage. For the process of uploading the user has to undergo the be encryption process with the encryption algorithm written in the system. Then the encrypted file gets stored in the cloud storage. Next process will occur if the user download or view the file on the storage because the encrypted file will be in the unreadable form where it has to undergo decryption as written on the system.

3.3 List of Modules

- User Module
  - Registration
  - Login
- Data Owner Module
  - Uploading File
  - Encrypting the uploaded file
- Data User Module
  - Decrypting the encrypted file
  - Downloading the File
- Storage Module - Cloud

3.4 Modules Explanation

- **User Module**

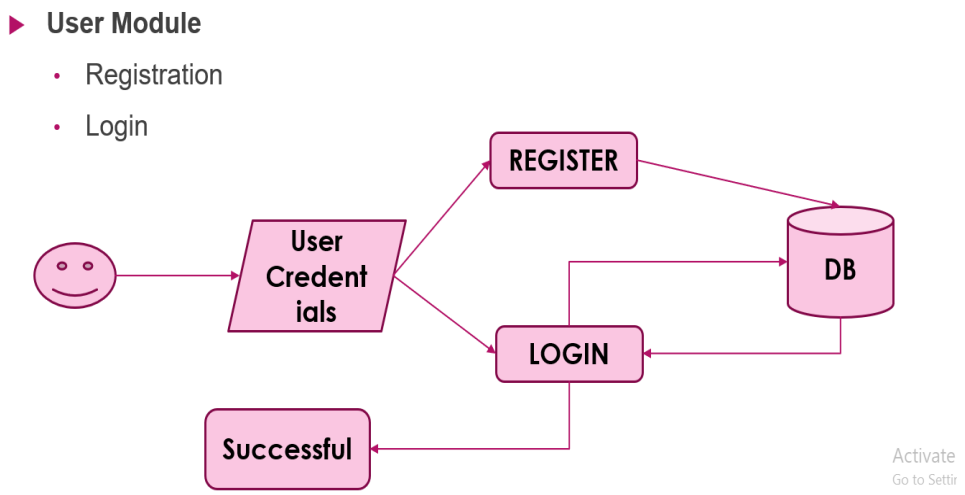

Figure (3.2) User Module

In the Registration part of the user module the UI gets the response of the user for the application's account registration. This credential gets stored in the Database. Then in the part of Login where the given user credentials for login is verified from the DB and after successful verification of the user the Home Page gets displayed.

- **Data Owner and Data User Module**

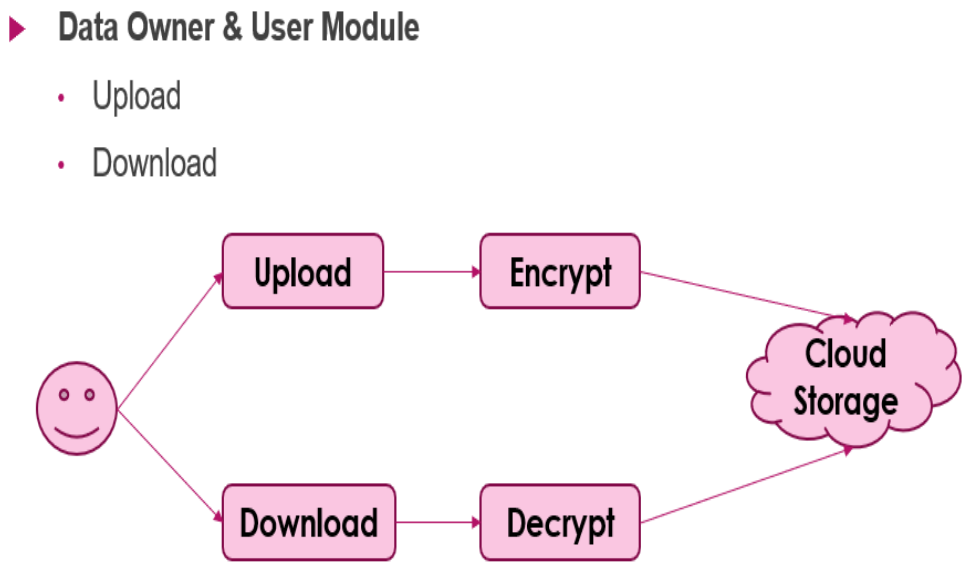

Figure (3.3) Data Owner and User Module

In the Data Owner module, we have to select the file that has to be uploaded and then that file will be encrypted with the hybrid algorithm of AES and RSA, where it will generate the key pair of public and private key for the encryption then the file is stored on the cloud platform.

In the Data User module, when the user selects the file to be downloaded or view then the user has to use their own private key for the decryption process, when the

decryption is successful then the file can be downloaded with appropriate details or the original file can be retrieved.

- **Storage Module**

  - Cloud Module

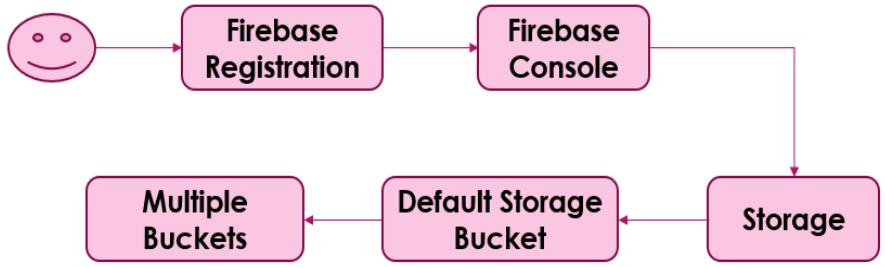

Figure (3.4) Storage Module

In the Cloud Storage module, the Firebase is considered for the storage purpose where Firebase has a feature named Storage, which is mainly for the storage of objects. The object can be of any type it can be image, video, text file etc., First the storage has to be prepared by sign-in into the Firebase console where it provides the Storage and then get started.

Then according to the application and the size of files storage, setting up your rules for public access.

Select a location for your default Cloud Storage bucket. This location setting is your project's default Google Cloud Platform (GCP) resource location. If we're not able to select a location, then your project already has default GCP resource location. It was set either during project creation or when setting up another service that requires a location setting. We can also create multiple buckets, each with its own location.

3.5 Algorithm Used

The algorithm used in this application to encrypt and decrypt the file is Hybrid Cryptography algorithm which is performed by combining AES and RSA algorithms.

The algorithmic steps that involved for the implementation process is

- **AES**
  - Step 1 : Substitution
  - Step 2 : Shifting rows
  - Step 3 : Mix columns
  - Step 4 : Add Round Key
- **RSA**
  - Step 1 : Generating keys → public & private key
  - Step 2 : Encryption → $C = P^e \bmod n$
  - Step 3 : Decryption → $P = C^d \bmod n$

CHAPTER IV

IMPLEMENTATION AND TESTING

4.1 Algorithms

The algorithm implemented here is AES (Advanced Encryption Standard), that utilize a same key for the sender and receiver and RSA (Rivest Shamir Adleman) utilizes totally unique keys for encoding and deciphering.

Here AES is symmetric cryptography and RSA is asymmetric cryptography. Symmetric cryptography is suitable for the encoding of an outsized measure of data. The AES calculation sketched out by the National Institute of Standards and Technology in US has been broadly acknowledged to switch DES in light of the fact that the new symmetric encryption calculation.

The AES calculation could be a symmetric square figure that procedures information pieces of 128bits utilizing a figure key of length 128, 192, or 256 bits. Each datum square contains a 4x4 exhibit of bytes called as the state, on that the AES calculation is performed. AES has 10, 12 and 14 rounds for 128-bit keys, 192-bit keys and 256-bit keys separately.

Keys	**No. of Rounds**
128	10
192	12
256	14

Table 4.1 Relationship between keys and no. of rounds

4.2 AES Algorithm

AES is based on a design principle known as a substitution-permutation network, and is efficient in both software and hardware. AES is the successor of DES, where DES use a Feistel network.

AES is a variant of Rijndael, with a fixed block size of 128bits, and a key size of 128, 192, or 256 bits. By contrast, Rijndael per se is specified with block and key sizes that may be any multiple of 32bits, with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4 x 4 column-major order array of bytes which is termed the state. Most AES calculations are done in a particular finite field. For instance, 16 bytes, $b_0, b_1, \ldots, b_{15}$ are represented as this two-dimensional array :

$$\begin{matrix} b0 & b4 & b8 & b12 \\ b1 & b5 & b9 & b13 \\ b2 & b6 & b10 & b14 \\ b3 & b7 & b11 & b15 \end{matrix}$$

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the plaintext and the final output, called the ciphertext. The number of rounds are as follows :

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

Description of the algorithm

1. KeyExpansion – round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.

2. Initial round key addition :
   - AddRoundKey – each byte of the state is combined with a byte of the round key using Bitwise XOR.

3. Rounds 9, 11, or 13 perform the following operations :
   - SubBytes – a non-linear substitution step where each byte is replaced with another according to a lookup table.
   - ShiftRows – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
   - MixColumns – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
   - AddRoundKey.

4. Final Round (total rounds 10, 12, or 14) :
   - SubBytes
   - ShiftRows
   - AddRoundKey

SubBytes

This step implements the substitution. In this step each byte is substituted by another byte. It is performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not

substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4) matrix. The next two steps implement the permutation.

ShiftRows

This step is just as it sounds. Each row is shifted a particular number of times.

- First row is not shifted
- Second row is shifted once to the left
- Third row is shifted twice to the left
- Fourth row is shifted thrice to the left

(Left circular shift is performed)

MixColumns

This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result. This step is skipped in the last round.

AddRoundKeys

Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128bits of data.

After all these rounds 128bits of encrypted data is given back as output. This process is repeated until all the data to encrypted undergoes this process.

4.3 AES Algorithm Implementation

Here in this application we are implementing the AES Encryption algorithm using ReactJS, for this we need CryptoJS library to be installed in our project using the command

- npm install crypto-js

Then we need that to be imported in our code to utilize the functions that are available in the package using the command

- import CryptoJS from “crypto-js”

To perform this encryption algorithm first we need to read the image file using the following code

```
<input type="file" id="image-file" />
file = document.getElementById('image-file').files[0];
```

Now the read file is converted into ArrayBuffer

```
let fread = new FileReader();

fread.onload = function(e){
   textread = e.target.result;
   enctext = UU.callEncrypt(textread);
}
fread.readAsArrayBuffer(file);
```

After ArrayBuffer this file converted into WordArray, and then it is converted to string, that is performed in the following code.
We need the string as input because the crypto-js library expects the input to be in the string format.

```
const CryptoJS = require("crypto-js");
function callEncrypt(argument){
   const wordArray = CryptoJS.lib.WordArray.create(argument);
   const str = CryptoJS.enc.Hex.stringify(wordArray);
   ct = CryptoJS.AES.encrypt(str, key);
   ctstr = ct.toString();
   return ctstr;
}
```

Then we create a Buffer from the encrypted string so that it can be uploaded into the storage bucket

```
let testBuffer = new Buffer(ctstr);
```

And then the bucket storage's index has to be returned.

For the purpose of decryption, the index that returned will help to download the file. The indexed file will be in the form of uint it has to be converted into string

```
var str = uintToString(file);
```

Next the decryption process is performed with the string that generated from the previous step, and it stored in the word array and then it has to converted to the bytearray

```
const decrypted = CryptoJS.AES.decrypt(str, key);
str = decrypted.toString(CryptoJS.enc.Utf8);
const wordArray = CryptoJS.enc.Hex.parse(str);
ByteText = wordArrayToByteArray(wordArray, wordArray.length);
```

Next the bytearray is converted to new uint8 array, now this array is used to create the blob file and save that into jpg file

```
var arraryBufferView = new Uint8Array(ByteText);
var blob = new Blob( [arraryBufferView], {type:"image/jpeg"} );
FileSaver.saveAs(blob,"decryptFile.jpg");
```

4.4 Other Implementations

Here is the starter code of this application (ie. App.js) which imports all the other pages namely Login, Register, Reset, Dashboard and the Route paths are mentioned in the code for navigation

App.js

```
import "./App.css";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Login from "./Login";
import Register from "./Register";
import Reset from "./Reset";
import Dashboard from "./Dashboard";
function App() {
 return (
  <div className="app">
   <Router>
    <Routes>
     <Route exact path="/" element={<Login />} />
     <Route exact path="/register" element={<Register />} />
     <Route exact path="/reset" element={<Reset />} />
     <Route exact path="/dashboard" element={<Dashboard />} />
    </Routes>
   </Router>
  </div>
```

); } export default App;

Registration.js

As mentioned above we have Registration Page which has input spaces for the user name, their e-mail address and password.

And also we have the Register button on submit which triggers the function of storing the registered user into the firebase users database.

We also have the Registration available with Google, where the authentication of Google Signin option in the Firebase console is enabled, so when the Register with Google button hits, it triggers the function of signInWithGoogle. And on the successful registration the page gets redirected to Dashboard page.

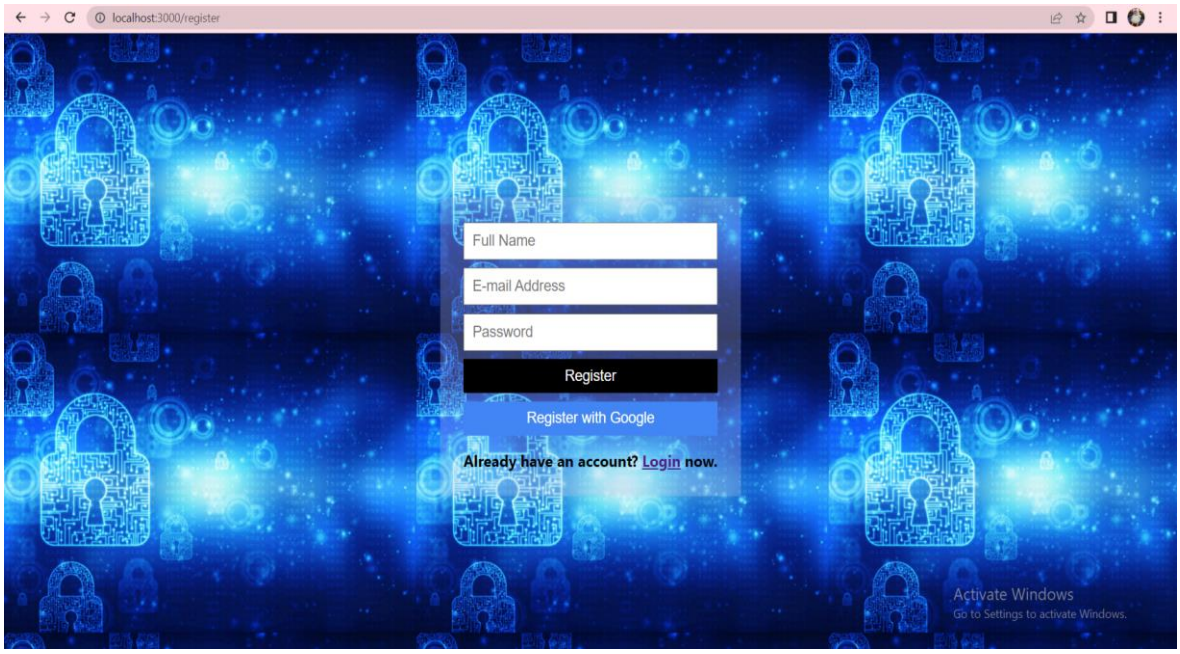

Figure (4.1) Registration Page

Registration TestCase

The basic testcase created in this registration part of the application is verifying the input of mail-id whether it is valid or not, where the user enters the input as simple invalid text in the space provided then it shows the error message as the input should be in the proper e-mail format.

```
const registerWithEmailAndPassword = async (name, email, password) => {
  try {
    const res = await createUserWithEmailAndPassword(auth, email, password);
    const user = res.user;
    await addDoc(collection(db, "users"), {
      uid: user.uid,
      name,
      authProvider: "local",
      email,
    });
  } catch (err) {
    console.error(err);
    alert(err.message);
  }
};
```

Figure (4.2) Registration Page TestCase Code

Testcase Output

Here the user hasn't given the mail-id in the proper required format, when the register button hits then it triggers the function to verify the given inputs and it detects the error in the email input, where it displays the error message as invalid email input

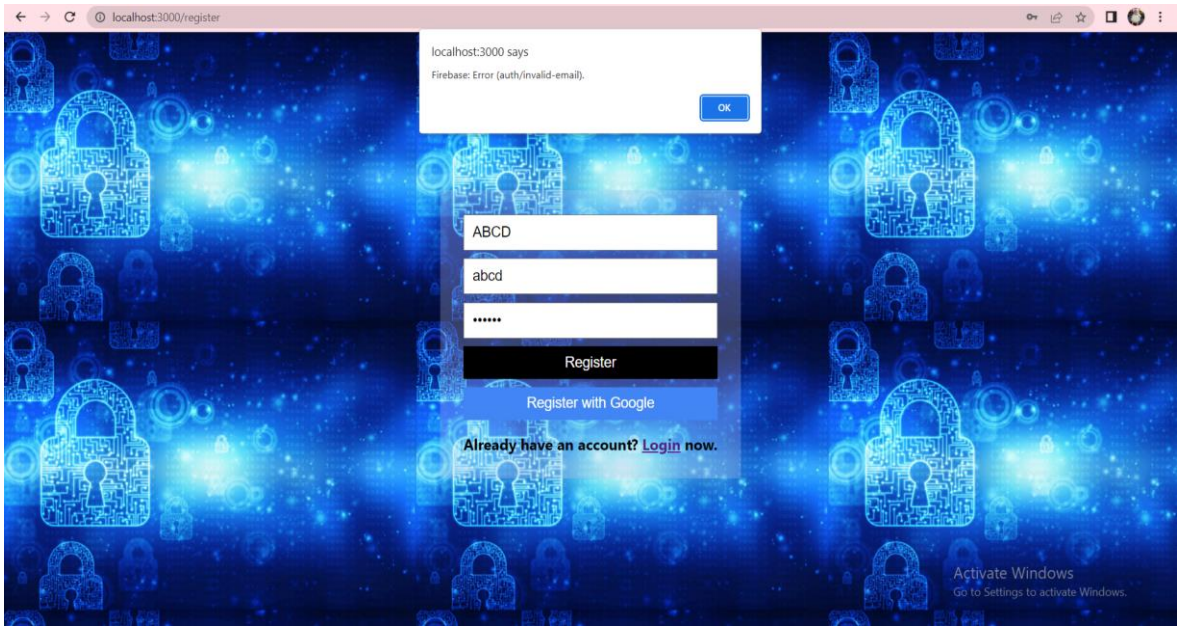

Figure (4.3) Registration Page TestCase Output

Dashboard.js

As same as Registration Page, we also have Dashboard which has inputs for the user to upload their file and also has the option to preview the file from the local.

And also we have the Logout button on submit which triggers the function to logout the current user that is logged-in.

We have the Upload button, which it triggers the function to upload the file into the Firebase Storage, and also we have the text mentioning the percentage of uploading the file into the storage bucket which is provided by the firebase for default storage.

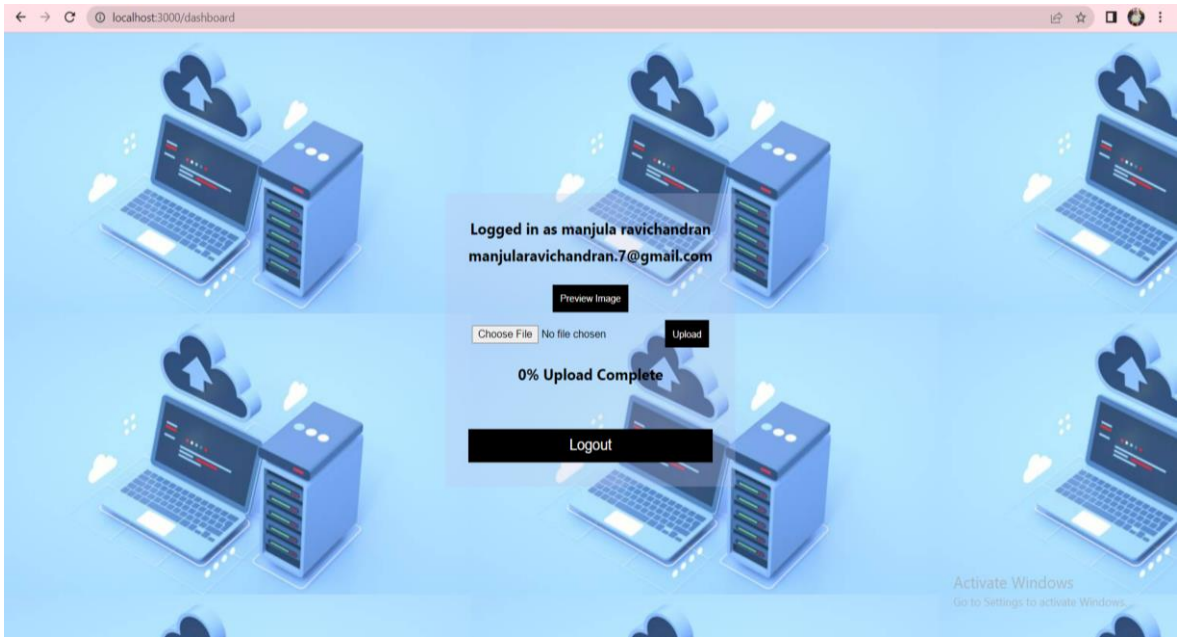

Figure (4.4) Dashboard Page

Dashboard TestCase

Here we have written the test case to be onClick of the upload button it triggers the function named handleUpload, which is used to verify if there is any file is selected by the user or not selected. If the function call doesn't pass any file as argument, which means there is no file selected, it displays the error message as "Please upload an image first".

```
function handleChange(event){
  setFile(event.target.files[0]);
}

const handleUpload = () => {
  if(!file){
    alert("Please upload an image first!");
  }

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

  uploadTask.on(
    "state_changed",
    (snapshot) => {
      const percent = Math.round(
        (snapshot.bytesTransferred / snapshot.totalBytes) * 100
      );
      setPercent(percent);
    },
    (err) => console.log(err),
```

Figure (4.5) Dashboard Page TestCase Code

TestCase Output

As mentioned above in the test case, here the user tries to click the upload button without any file selected, so it displays the message to select an image first

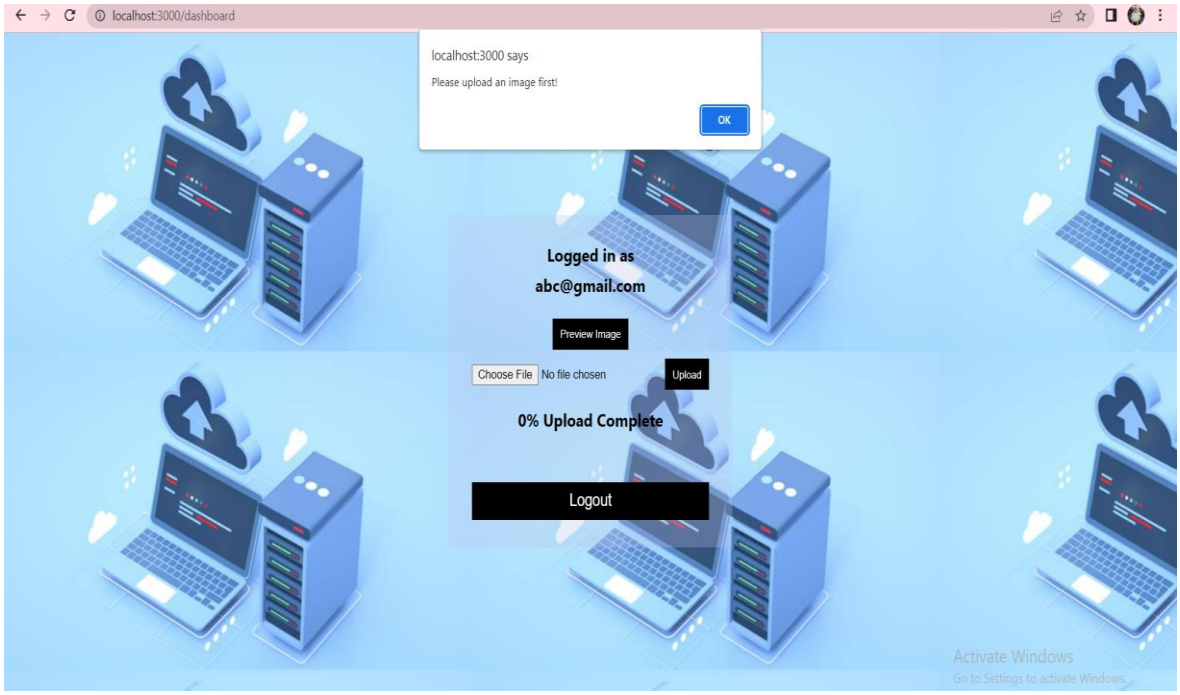

Figure (4.6) Dashboard Page TestCase Output

Firebase Storage Rules Configuration

For utilizing the Storage menu that is available in the Firebase, firstly we have to change the rules with the option edit rules, (Figure 4.7) where slight modification in the rule has to be done, that allow read, write: if has to be setted as true, which will enable us the option of writing the file and reading the existing file from the Storage, which will provide the (Figure 4.8) default storage bucket for the user.

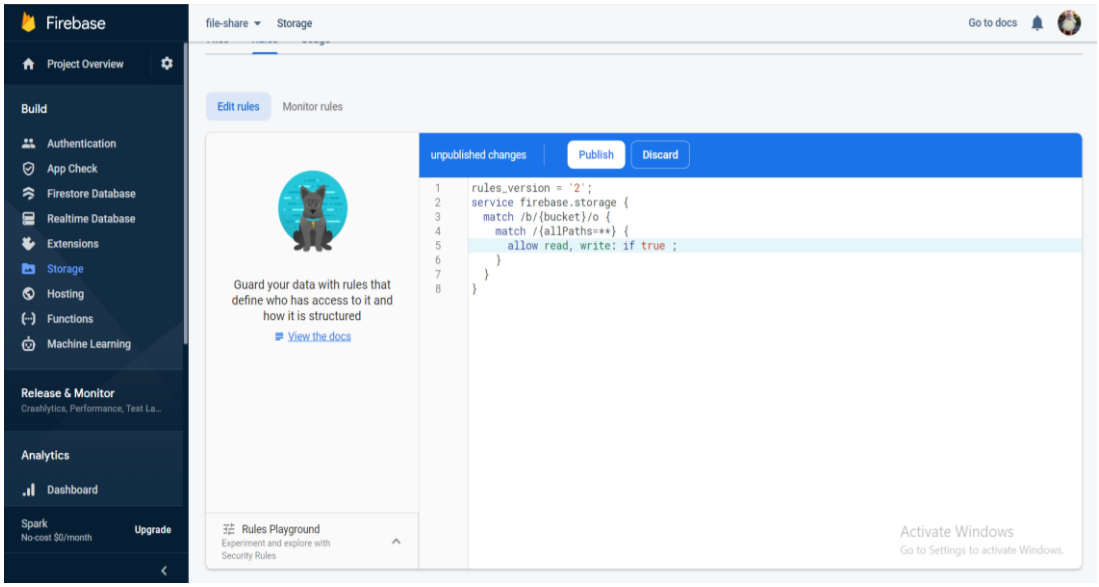

Figure (4.7) Setting rules for Firebase storage

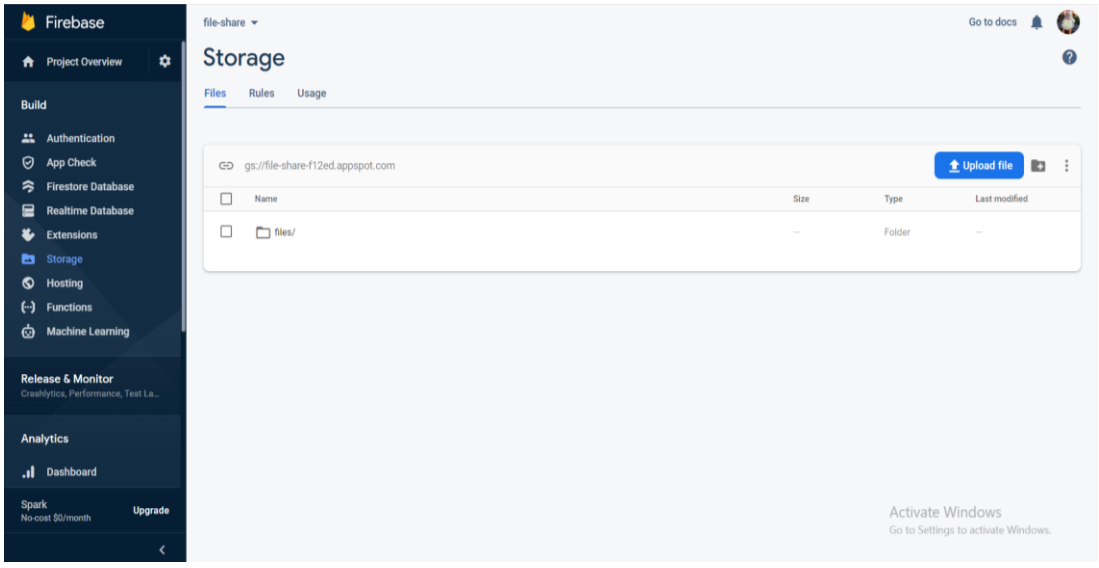

Figure (4.8) Default storage bucket

Multiple Bucket Configuration

Apart from Default Bucket Storage, we can also create multiple buckets in the Firebase Storage with the help of Blaze, where it has the pricing details, after the payment process then the access for multiple bucket creation is enabled for the user. So it can create multiple buckets for different purpose of storing the files and documents in the storage.

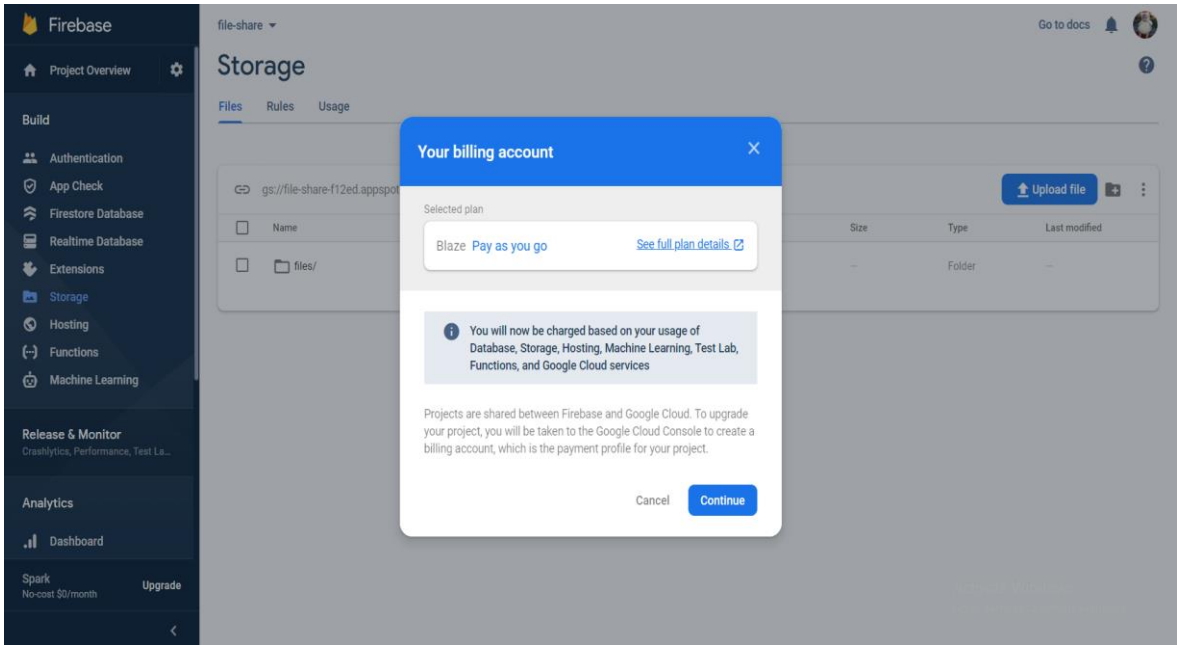

Figure (4.9) Example for Multiple Bucket Configuration

REFERENCES

[1] Mr. Uttam Kumar & Mr. Jay Prakash, **"File Storage on Cloud"**, International Journal of Creative Research Thoughts (IJCRT), Volume 8, Issue 7 July 2020.

[2] M. Arumugam, et al, **"Secure data sharing for mobile cloud computing using RSA"**, IOP Conference Series: Materials Science and Engineering, 2021.

[3] Mrs. Soja Salim, **"SECRY – Secure file storage on cloud"**,

APJ Abdul Kalam Technological University, Issue 11 July 2020.

[4] Shruti Kanatt, et al, **"Review of Secure File Storage on Cloud"**, International Journal of Engineering Research & Technology (IJERT), Volume 9, Issue 2 February 2020.

[5] Dr. M. Naveetha Krishnan, Mr. T. Tamilarasan, **"Secure File Storage on Cloud using Hybrid Cryptography"**, International Journal of Advanced Research in Computer Science Engineering and Information Technology (isrjournals), Volume 6, Issue 1 April 2021.