



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

**PARALLEL AND DISTRIBUTED COMPUTING
(CSE4001)**

FALL SEMESTER 2021 – 2022

PROJECT REPORT

**DECENTRALIZED CLOUD STORAGE SYSTEM
USING BLOCKCHAIN**

TEAM MEMBERS

B. ARAVINDA – 19BCE1190

P. K. AMUDHINI – 19BCE1492

S. DIVYASHREE - 19BCE1689

FACULTY

DR. GAYATHRI. R

I. ABSTRACT

Cloud storage has become a prominent study area as a result of the vast amount of data produced every day. The current centralised cloud storage system has several disadvantages, including high operational costs, data availability, and data security. Blockchain presents a decentralised approach to resolving such issues. By combining the cloud and blockchain technologies, the storage solution is made secure and scalable. The major goal is to create a system that takes advantage of the security of cloud-based data and applies it to the blockchain. It enables users to save data to the cloud and includes a conspicuous access control mechanism to maintain data privacy. Without valid permission, no third party will be able to view the data since it is encrypted and stored in distinct nodes. This will improve the security of existing cloud storage and reduce data breaches and other assaults. This project proposes a blockchain-based cloud storage system in which data is divided into multiple pieces that are encrypted and linked using a hashing mechanism. The IPFS (Interplanetary File System) protocol distributes file portions throughout a peer-to-peer network of computers, allowing for decentralised storage of the file.

KEYWORDS *Cloud Storage; Blockchain; IPFS; AES Encryption*

II. LITERATURE REVIEW

Paper - Study on Data Security Policy Based on Cloud Storage

The user services provided by cloud computing systems, such as high-speed storage and retrieval, are critical in the cloud computing environment. Meanwhile, data security is a critical issue that cloud storage technologies must address quickly. Malicious assaults on cloud storage systems have increased in recent years, and data leakage from cloud storage systems has also become increasingly common. Cloud storage security is concerned with the safety of the user's data. The goal of this work is to achieve cloud storage data security and to develop a cloud storage security policy to go with it. By examining the security concerns of user data in cloud storage and approaching a subject of relevant security technology based on the structural characteristics of the cloud storage system, these were merged with the results of existing academic research. Instead of being saved on the cloud service provider's equipment, the user's data could be stored in a shared cloud storage system.

As a result, users have no control over data security to ensure confidentiality and integrity. This paper summarises a perfect security policy that applies to cloud service providers and users and aims to ensure the safety and reliability of data as well as the benefits of cloud service providers and users to attract more Internet users and successfully promote cloud computing and cloud storage systems. In this work, a few technical issues related to cloud storage security are examined from a technical standpoint. Academia, industry, and government departments must collaborate to achieve total cloud storage security.

Paper - Decentralizing Privacy: Using Blockchain to Protect Personal Data

Users own and control their data thanks to a decentralised personal data management system. The authors have developed a technology that transforms a blockchain into an autonomous access-control manager that does not rely on third-party trust. Unlike Bitcoin, their system's transactions aren't strictly financial; they're used to carry commands like saving, querying, and sharing data. Finally, they addressed the future blockchain additions that could turn them into a well-rounded answer for society's trusted computing concerns. They addressed the privacy concerns that users have when using third-party services throughout this article. They concentrated on mobile platforms, where services distribute applications for customers to download. These applications are constantly collecting high-resolution personal data over which the user has no control. They felt the services were trustworthy but curious.

It's worth noting that the same method might be used for other data privacy problems, such as people sharing their medical data for scientific research, with the opportunity to monitor how it's used and opt-out at any time. As a result, the system guards against challenges such as Data Ownership, Data Transparency and Auditability, and Fine-grained Access Control. Users are not obligated to trust third parties and are always informed of the data collected about them and how it is used. Furthermore, the blockchain acknowledges users as the owners of their personal information. As a result, businesses can concentrate on using data rather than worrying about properly safeguarding and compartmentalizing it.

Paper - Meta-Key: A Secure Data-Sharing Protocol Under Blockchain-Based Decentralized Storage Architecture

The authors offer Meta-key, a data-sharing mechanism that allows users to share encrypted data in a decentralised storage architecture based on blockchain. The owner's public

key encrypts all data-encryption keys, which are then stored on the blockchain for safe and secure storage and key management. To provide secure data exchange in an untrusted environment, encrypted data is stored on dedicated storage nodes and a proxy re-encryption technique is implemented. The model's security study reveals that, due to the Meta-unique key's architecture, the proxy re-encryption used in the system is inherently free of collusion-attack. The security of data is secured in a blockchain-based cloud storage system through encryption and the concealment of the location.

Because these locations are stored in meta-data that can only be viewed by data owners, they employ a new decryption key and location for the shared data that is distinct from the original copy and only known to the receiver to ensure that both parties' data is secure after sharing. Requesting that the data-owner download, decrypt, re-encrypt (with a new decryption key), and upload again (to a different place) is not only wasteful but also creates security problems because the plain text must be recovered during the sharing process. Proxy re-encryption is chosen as the cornerstone of the key and cipher-text transformation process to address these issues. They presented a Meta-key-based technique for secure data exchange in a blockchain-based decentralised storage system. They concentrated on the suggested cryptographic protocol's collusion-free characteristic and rigorously demonstrated it.

Paper - A Next-Generation Smart Contract and Decentralized blockchain Platform: A case study on Ethiopia

The distinction between the traditional approach to contracts and smart contracts is discussed in this paper. In the corporate world, smart contracts are available, but they are unsecured. Smart contracts are employed in this article using fresh emergent blockchain technology. Bitcoin is a cryptocurrency that is used for transactions. By incorporating blockchain technology into smart contracts, the process will become faster, more dependable, secure, and trustworthy. The use of blockchain technology will improve the platform for small-scale contracts, potentially lowering broker fees. All of the transactions are transparent and secure. The research proves that secure business transactions in a decentralised system are possible.

The new revolutionary technology of Blockchain, as well as its use in E-commerce, is regarded as a powerful instrument. It fascinates newcomers, and it fosters a trustworthy environment and secure transactions. In this article, a smart contract application for business-

to-government (B2G) is studied for the evaluation of the blockchain and bitcoin for smart contracts. When blockchain is applied to government contract bids, the contractor will have access to information from the beginning to the end. Three smart contracts are linked to government business operations (registration, tax authorities, banks, Anticorruption Bureau, and so on) in this approach. The blockchain revolution is predicted to be a viable or complementary alternative to the current system. It promotes small contractors to do their work without using brokers, allowing them to save money.

The proposed approach blends the idea of a B2G with blockchain technology to create an effective, trust-based environment for this sort of contract's success and to alleviate consumer concerns. This concept builds confidence between contracts and gives them more options for accessing their information at any time. Another benefit is that once they develop a document, they may reuse it for additional contract applications, saving time and ensuring that the documents are not tampered with. It will lessen the anxiety among the contractors. Blockchain technology has a wide range of applications. It can be used for bank transactions, ECommerce (C2C, B2B, C2B, and so on), educational purposes (to save educational papers), financial services (import and export), and insurance.

This study makes a significant contribution to the use of advanced technology and E-commerce, which requires practical research for B2C to cover various aspects of smart contract operations that aid in the conversion of traditional operations to digital operations. All of the country's operations will be digitalized and kept safely if Blockchain technology is used.

Paper - Blockchain-based Decentralized Storage Scheme

This paper offers a blockchain-based decentralised storage system that can make full use of the leftover space on users' hard discs all over the world. The storage provider issues a data integrity certificate to the user, and the user pays the storage cost to the storage provider via lightning network technology after certifying that the verification was successful. The blockchain stores all proofs and payment information, ensuring the system's security and trustworthiness. This scheme has been improved in terms of system access and payment mechanisms when compared to current mainstream distributed storage systems. The user delivers the encrypted data to the middleman, who sends it to the storage provider and notifies the user of its position. The user pays the storage cost to the storage provider using lightning

network technology when the data integrity certificate between the user and the storage provider is finished.

The authors have presented a distributed storage method based on blockchain technology, as well as a detailed description of the system design. Blockchain technology, lightning network technology, remote data integrity certification, and remote data confidentiality protection technology are all used in the system. Make the most of the user's remaining disc space, eliminate resource waste, and encourage nodes to sign up for the cloud storage service. The system proposes a storage protocol with a middleman that can handle system updates, upgrades, and access, among other things.

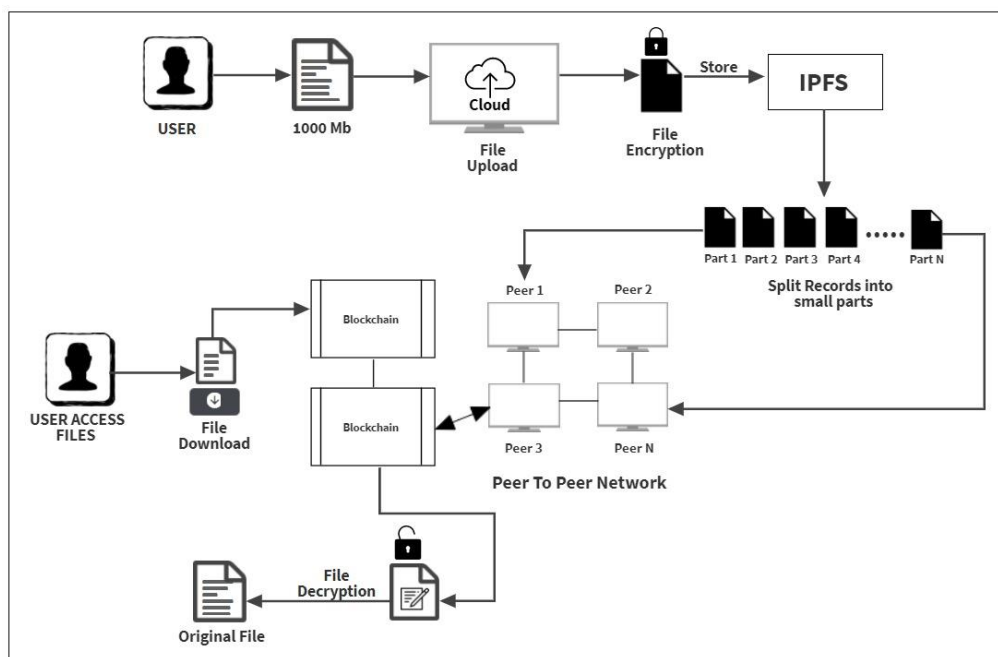
III. EXISTING SYSTEM

Cloud computing refers to a centralized system in which the service provider has total control over the data kept in their cloud. Providers of centralized cloud storage services demand a high fee for their services. While these costs do not seem significant to bigger businesses, they may have a significant impact on smaller businesses. The majority of contemporary trends, such as Amazon, Microsoft, and IBM, employ cloud-based systems for data storage, which are single-node/point systems. The core idea is to exchange a file over a peer-to-peer (P2P) network and use a blockchain to track the encryption, storage, retrieval, and contract administration. Data is decentralized on blockchain. To guarantee redundancy, the file is then broken into multiple pieces and duplicates of each component are made. The data is encrypted and kept on many nodes, preventing it from being hacked by a third party.

IV. PROPOSED ARCHITECTURE

The proposed system is divided into four sections. The user must first create a MetaMask account. The user's account address and wallet balance are retrieved from the MetaMask using web3.js in the app. The uploaded file is encrypted using the AES method. The IPFS protocol is used by the system to effectively distribute files among multiple peers in the network. The IPFS then provides a hash value containing the file's path. The Ethereum blockchain network includes smart contracts, which allow users to keep information about files they upload on the blockchain. When a user requests a file, IPFS is fetched from the blockchain and the file is downloaded from the peer-to-peer network. Every time data is uploaded or downloaded, the proposed system encrypts and decrypts it. Thus, the original file is obtained.

Smart contracts are implemented in the proposed system to store file data in the blockchain as well as transfer cryptocurrency from the user's wallet to the peer's wallet. The AES encryption algorithm is used to improve the security of data stored in cloud storage by users. Only the legitimate owner of the file can access the data, according to the proposed system, that connects the user's wallet address to the user's data. The Ethereum blockchain stores the data of users. The Ethereum blockchain supports smart contracts, which allow users to keep information about files they upload on the blockchain.



V. MODULES

1. Login / Sign up:

A password-less login concept is proposed in this project. The users are identified based on their MetaMask address. Signature analysis is done to verify the account holder. The basic idea is that it's cryptographically easy to prove the ownership of an account by signing a piece of data using a private key. Therefore a message-signing-based authentication mechanism is built with a user's public address as their identifier. During the Signature analysis, a One-Time Nonce (OTN) is performed, which generates a random number to authenticate the account. With the nonce, public address, and signature, it is possible to cryptographically verify that the nonce was signed correctly by the user.

2. File Upload:

In this module, the files are uploaded to the IPFS. The IPFS or InterPlanetary File System is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. The IPFS splits the file into pieces and sends it to different nodes across the peer-to-peer (P2P) network. The pieces are replicated on multiple nodes for the high availability of the file. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

The application encrypts the file before uploading it to the IPFS. Every file is encrypted using AES-256 symmetric encryption. Advanced Encryption Standard (AES) is one of the most frequently used and most secure encryption algorithms available today. AES uses the same secret key (passphrase) for encryption and decryption. The user provides the passphrase to encrypt the file, thereby protecting the confidentiality of the user's data.

3. Blockchain / Smart Contracts:

A blockchain is a database that stores data blocks and links them together to build a chronological single-source of truth for the data. This project makes use of the Ethereum blockchain. Ethereum is a decentralized, open-source blockchain with smart contract functionality. Smart contracts are essentially programmes stored in the blockchain that execute when certain criteria. They're usually used to automate the execution of an agreement so that all parties may be confident of the conclusion right away, without the need for any intermediaries or time waste. The smart contracts are created for storing the login details and the IPFS hash.

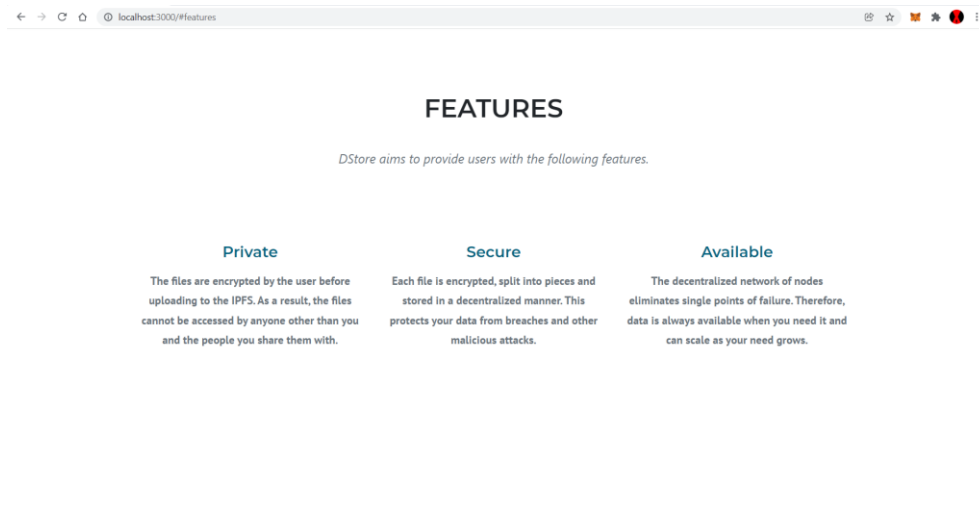
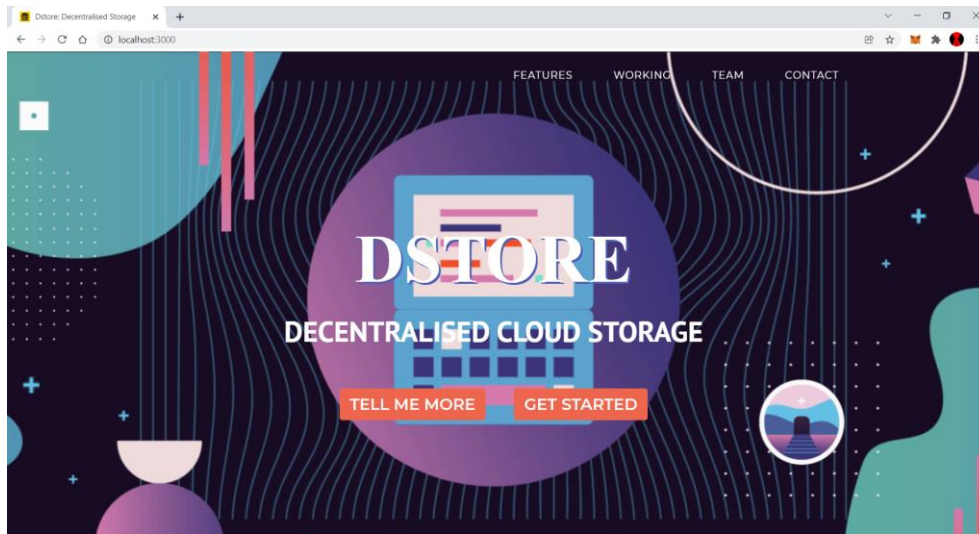
4. Web application:

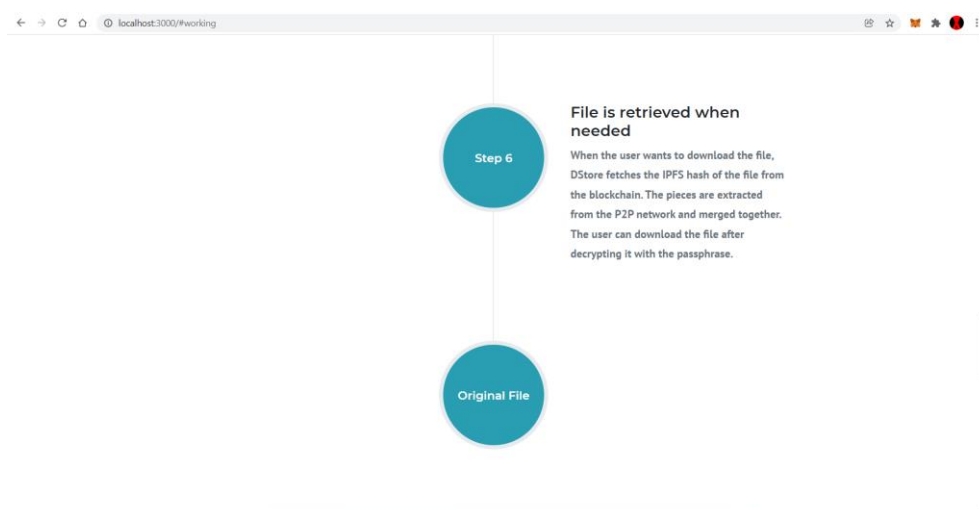
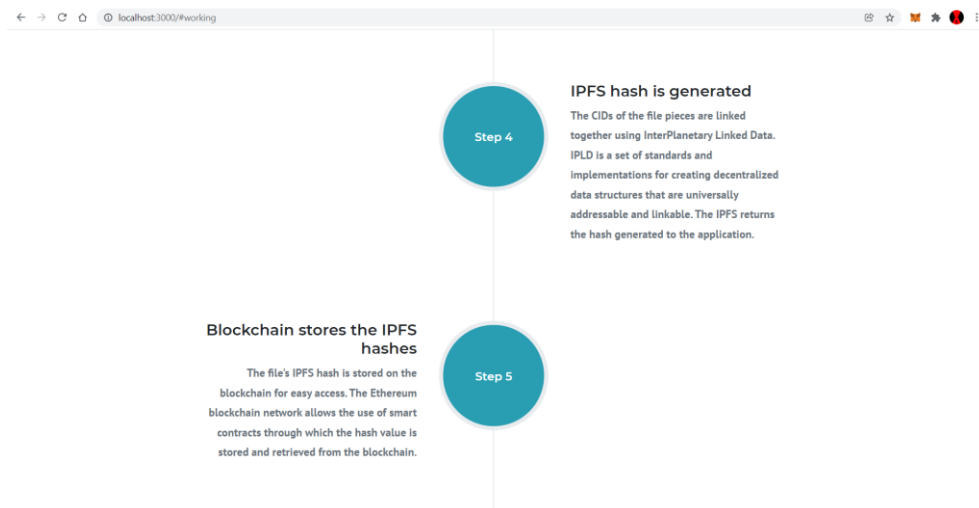
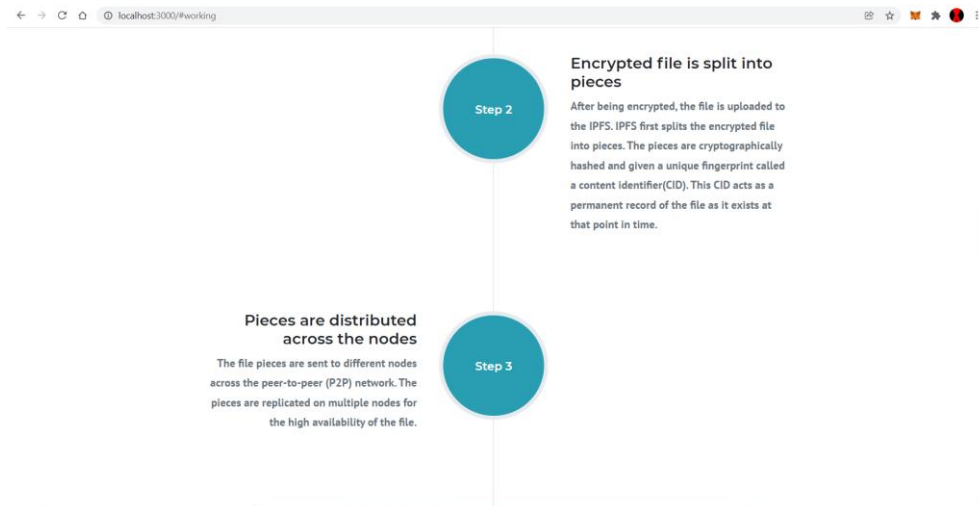
The project uses React.js to create a web application for the proposed system. Various web pages are created which provides different functionalities.

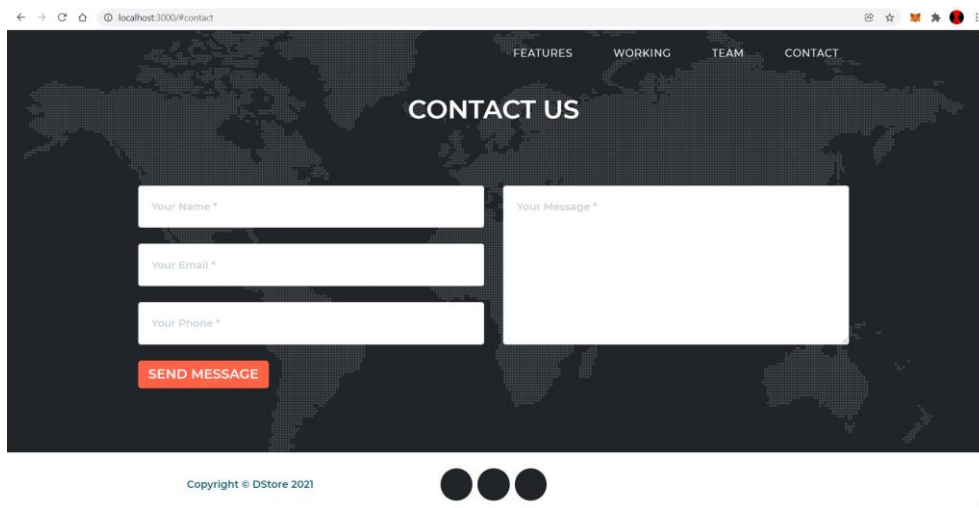
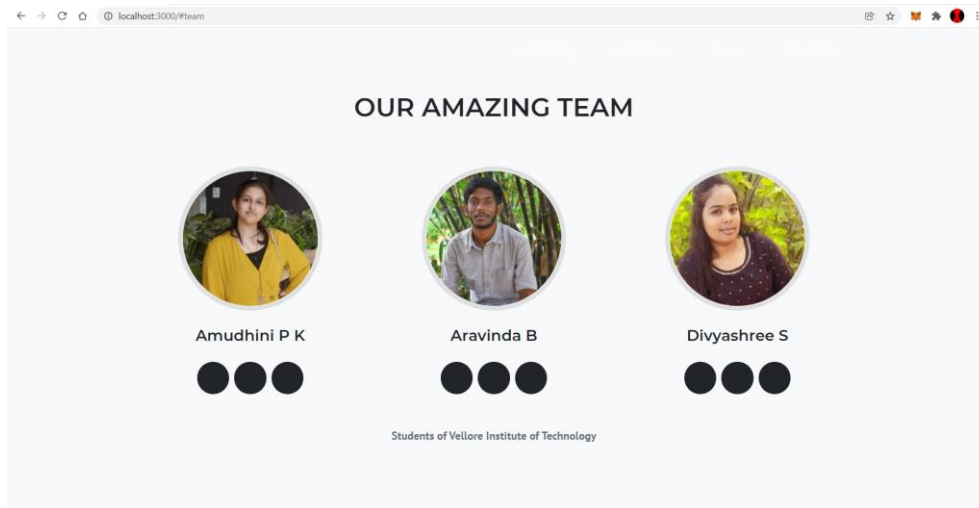
- i. **Home page** - provides detailed information about the application for new users
- ii. **Login page** - necessary for user identification and authentication
- iii. **Dashboard page** - displays user information and provides a summary regarding the files uploaded till now
- iv. **File upload page** - The file upload, download and sharing feature is available in this page
- v. **Contact page** - the user can report any issues faced in the application

VI. APPLICATION SCREENSHOTS

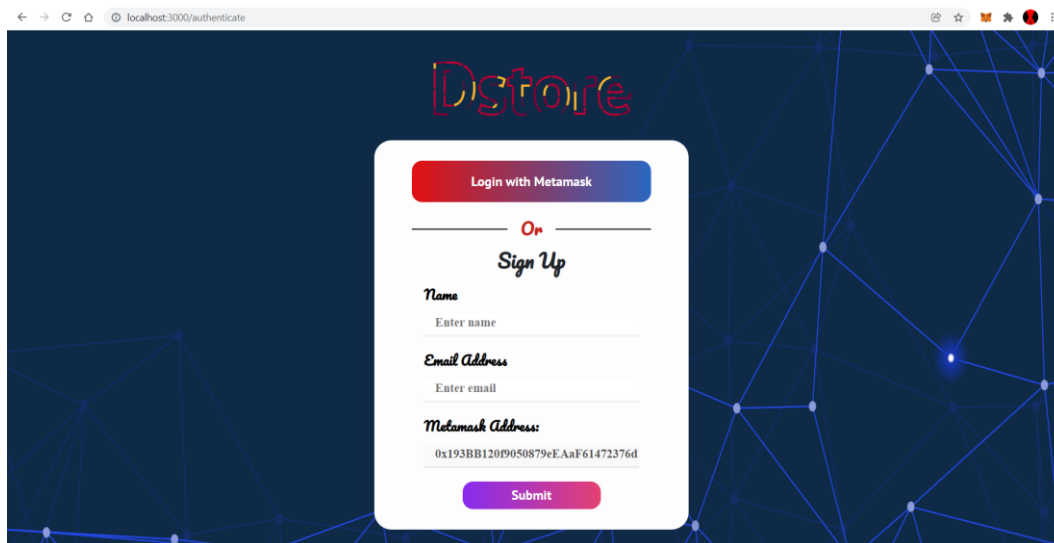
i. Home Page:





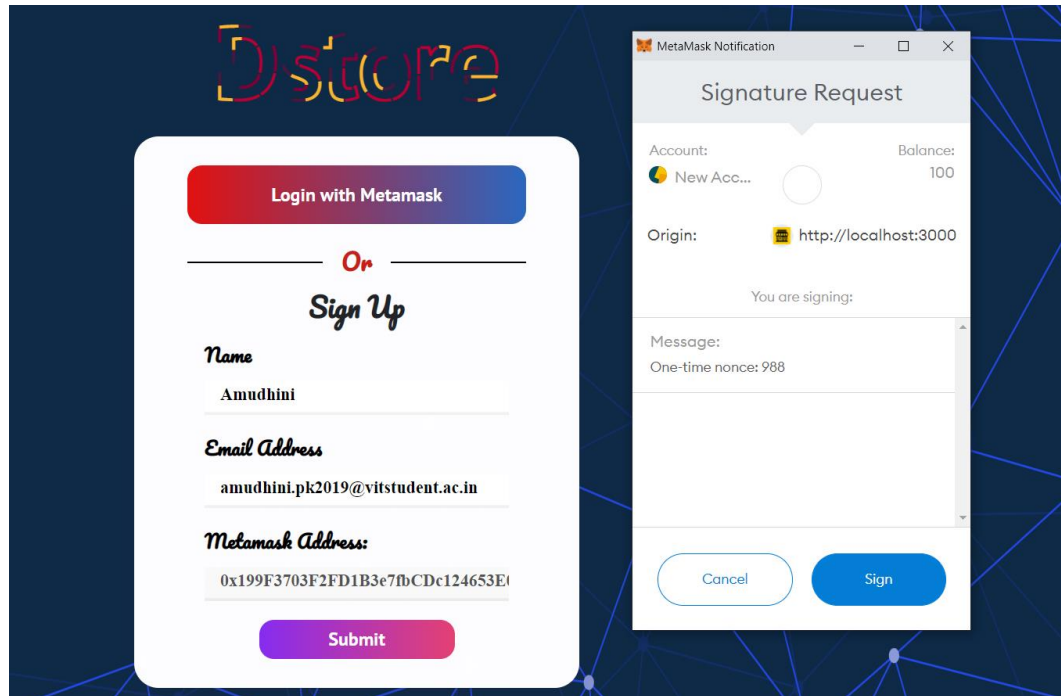


ii. Login Page:

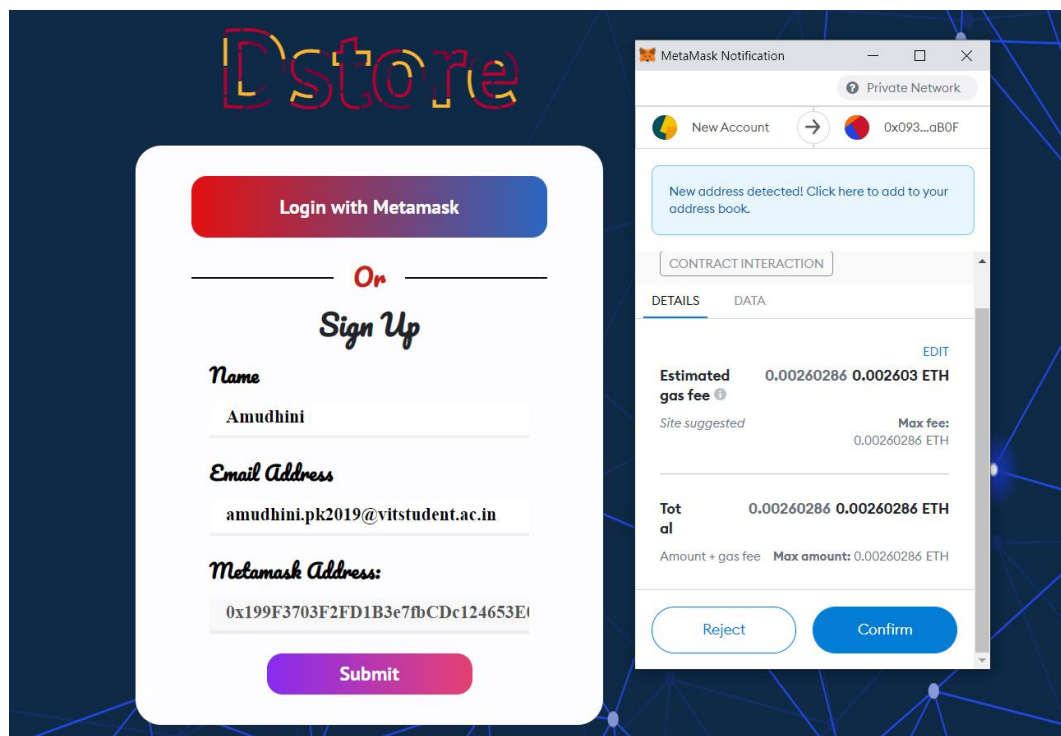


New user sign up:

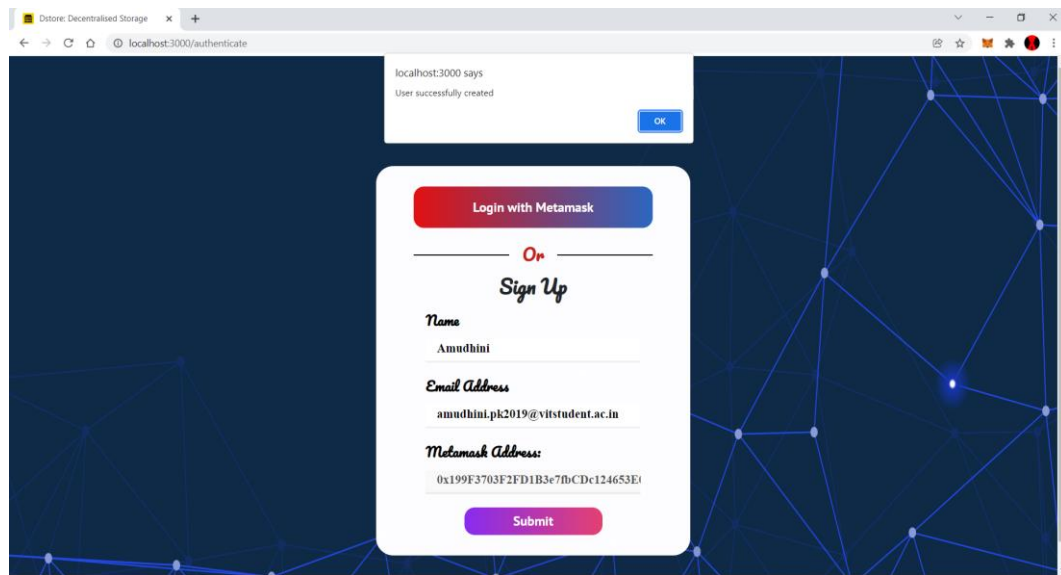
- The user enters the required details and clicks submit.
- Then signature request is sent to the user along with the One-Time Nonce which the user has to sign.



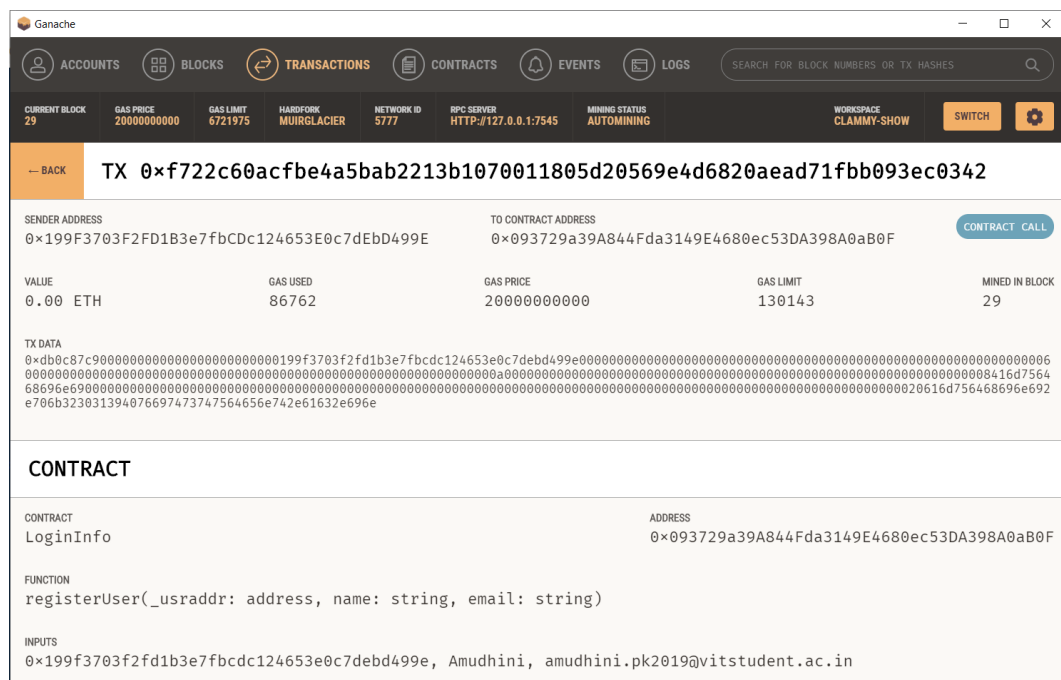
- After signature analysis, the user is asked to pay the gas fee to create an account. This gas fee is for storing the login information in the blockchain.



→ After the transaction occurred, the user account is successfully created.

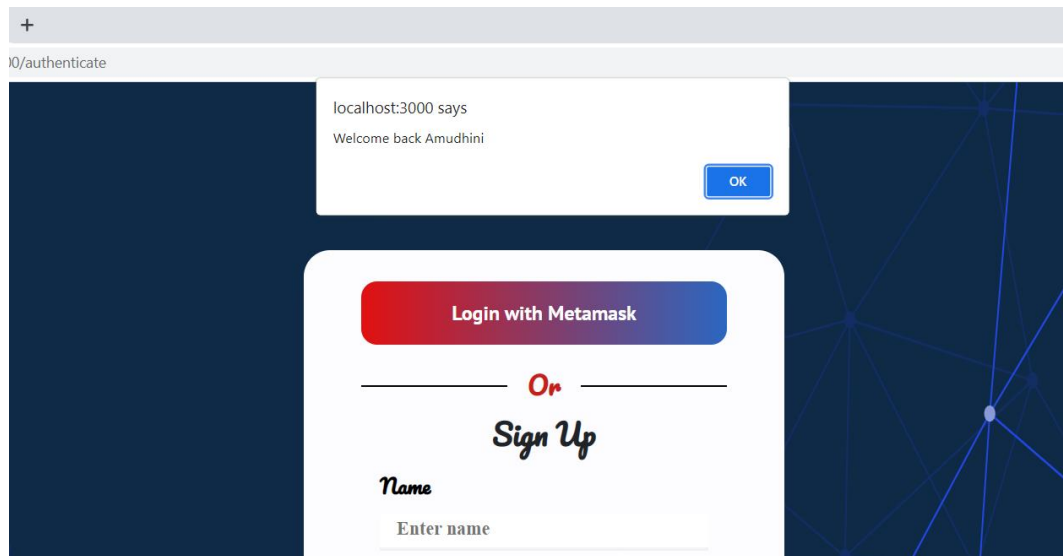


→ The transaction can be viewed from the Ganache. It is observed that the transaction happened between the sender and the contract deployed in the blockchain.



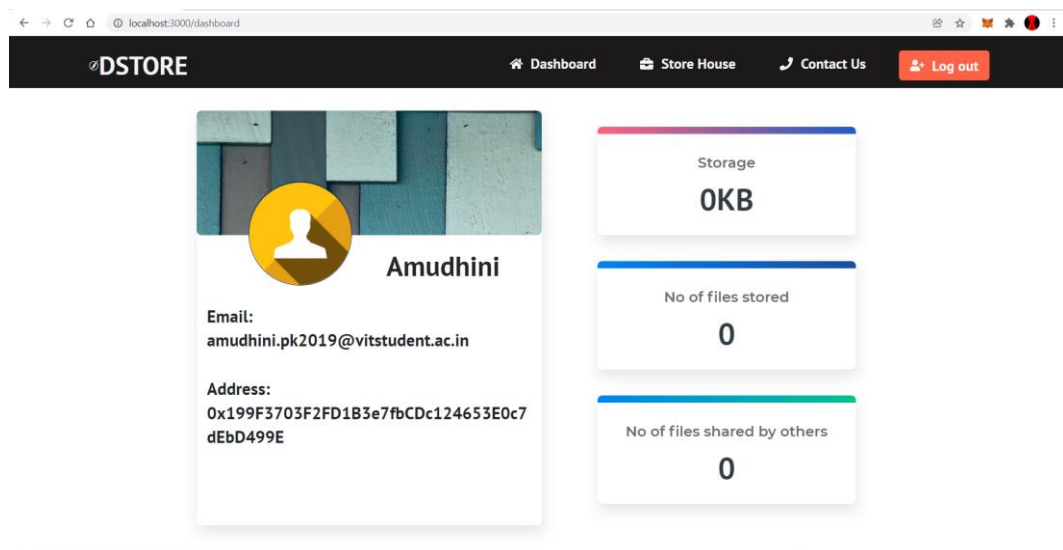
Existing user sign up:

- In the login page, when an existing user clicks “Login with Metamask”, request for signature analysis is first sent.
- After that the application checks for the user, if the user account exists, a message pops up saying “Welcome back -user-”.
- Then the application redirects to dashboard.

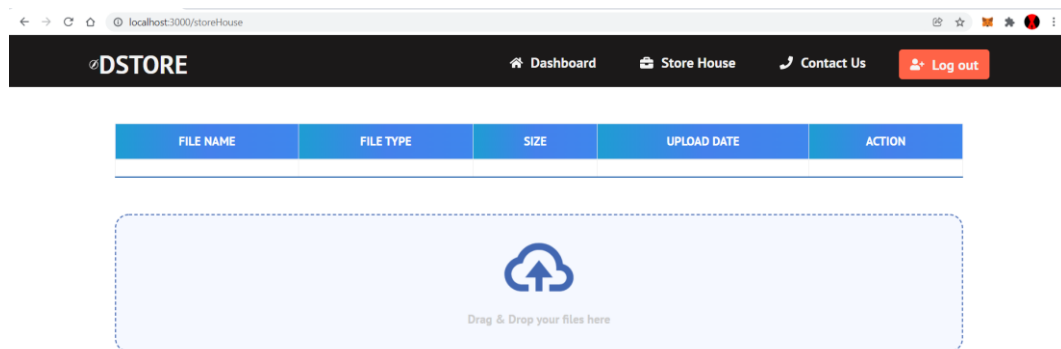


iii. Dashboard:

- The dashboard provides summary of the user information.
- The user's name, email and metamask address is displayed in the left box.
- The right side has 3 boxes providing different information regarding the files uploaded:
 - **Storage** which displays how much the user has uploaded in the application in KB.
 - **No of files stored**
 - **No of files shared by others**

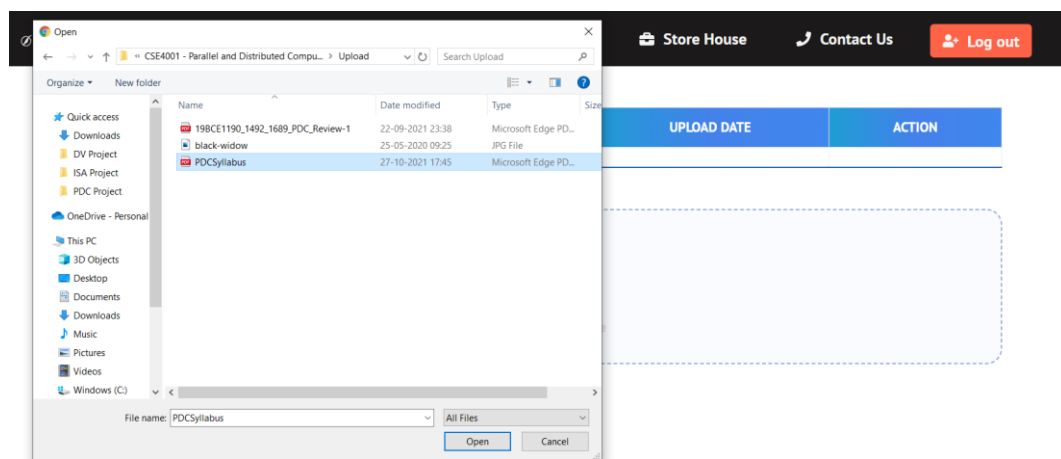


iv. File Upload:

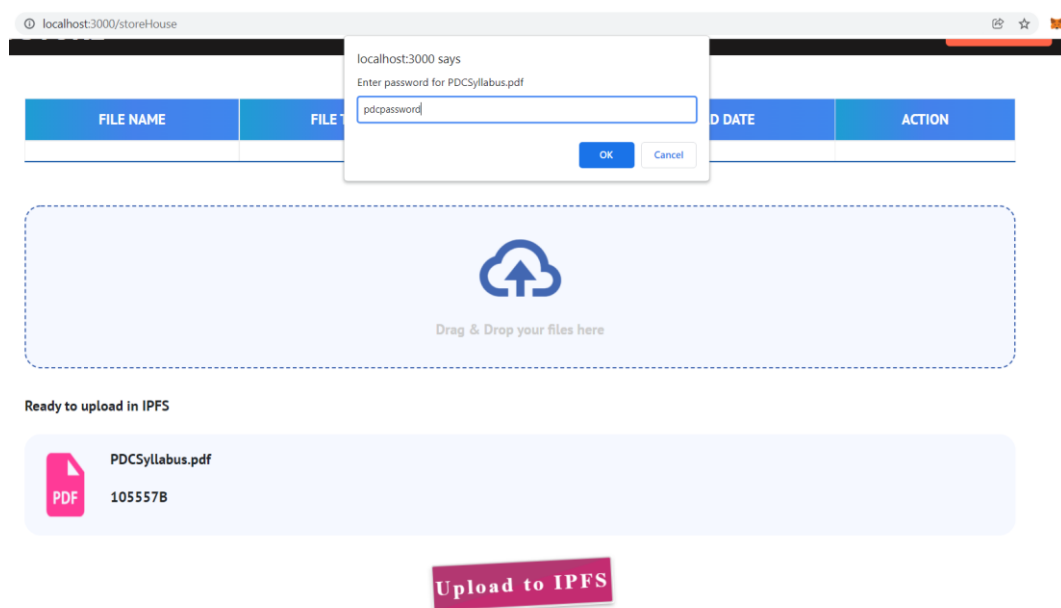


Uploading a file in the application:

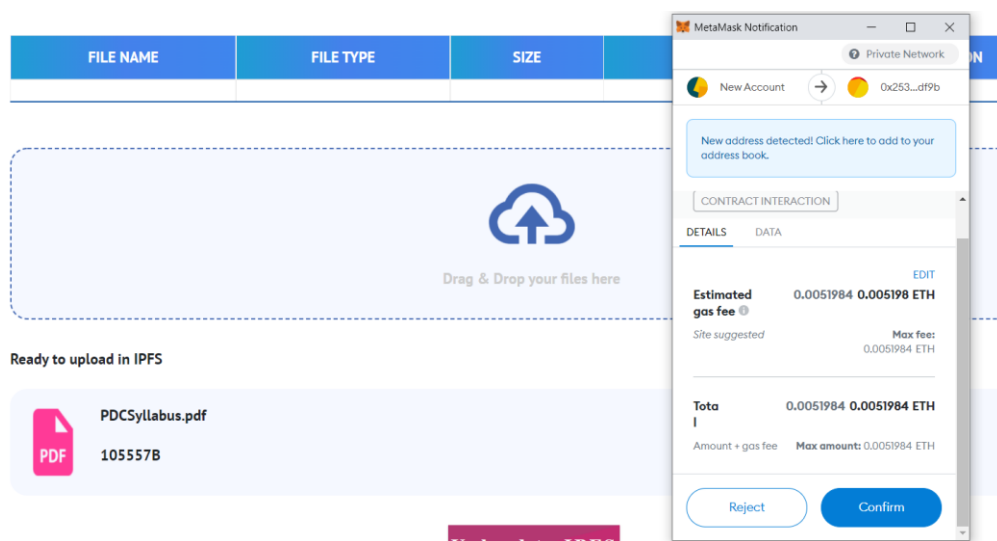
→ The user selects the file to be uploaded.



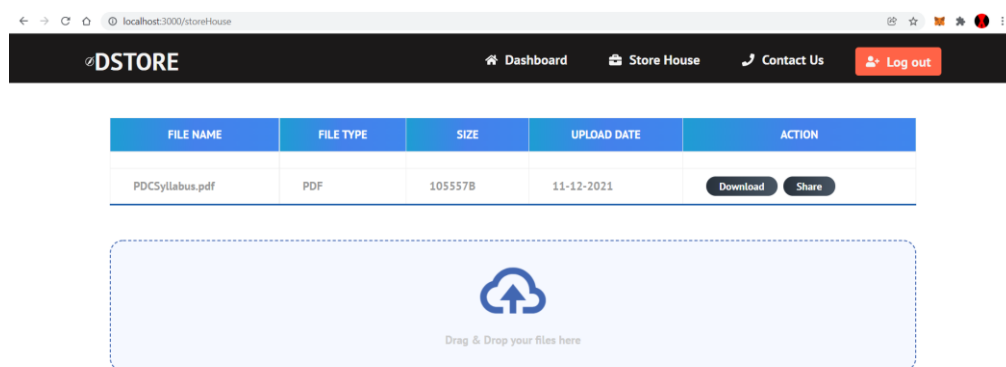
→ Then the user enters the password to encrypt the file.



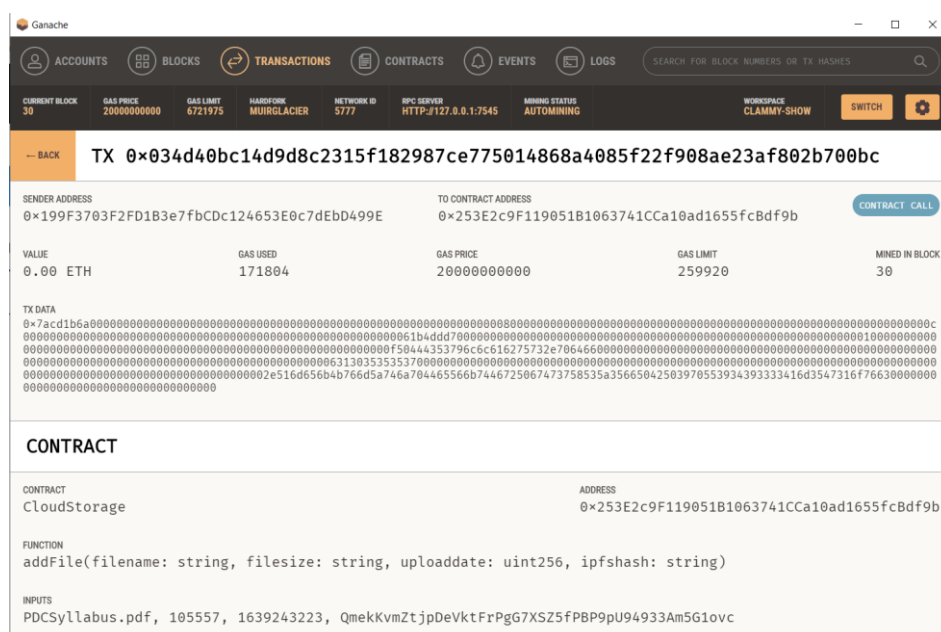
- The file then will be uploaded in the IPFS and a hash value will be sent back which should be stored in the blockchain.
- A transaction occurs where the user pays a gas fee.



- After successful transaction, the file is displayed in the table.

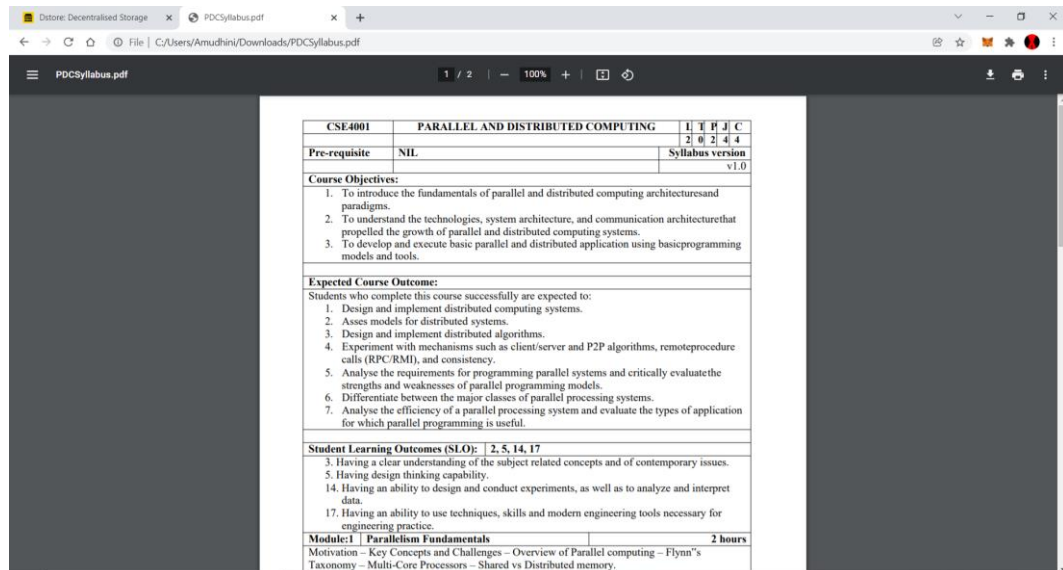
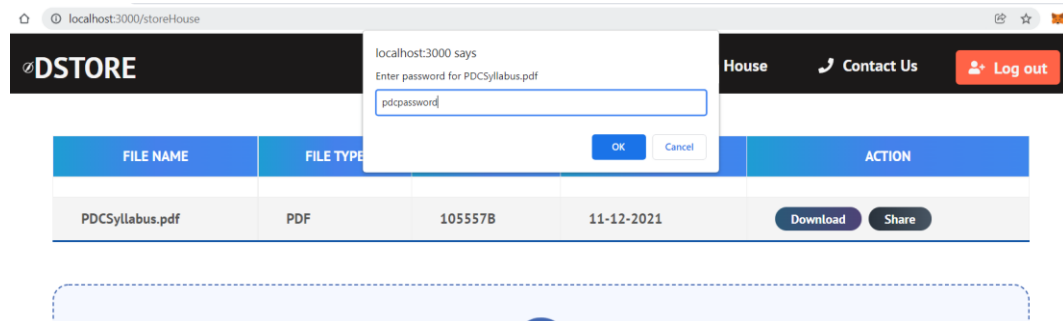


- The transaction can be viewed in the ganache as shown below.

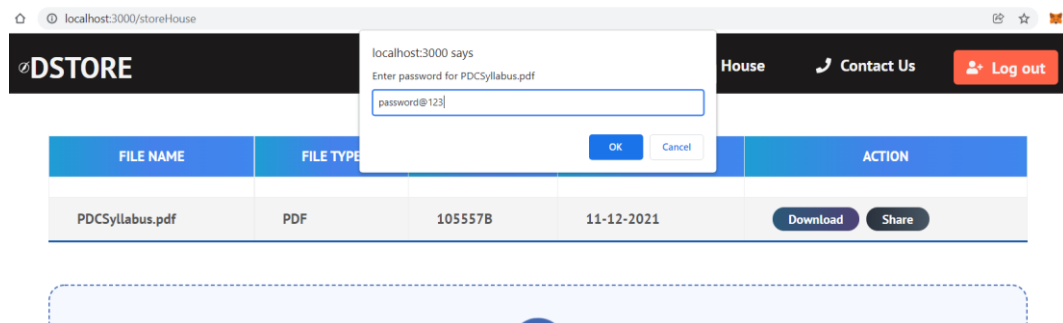


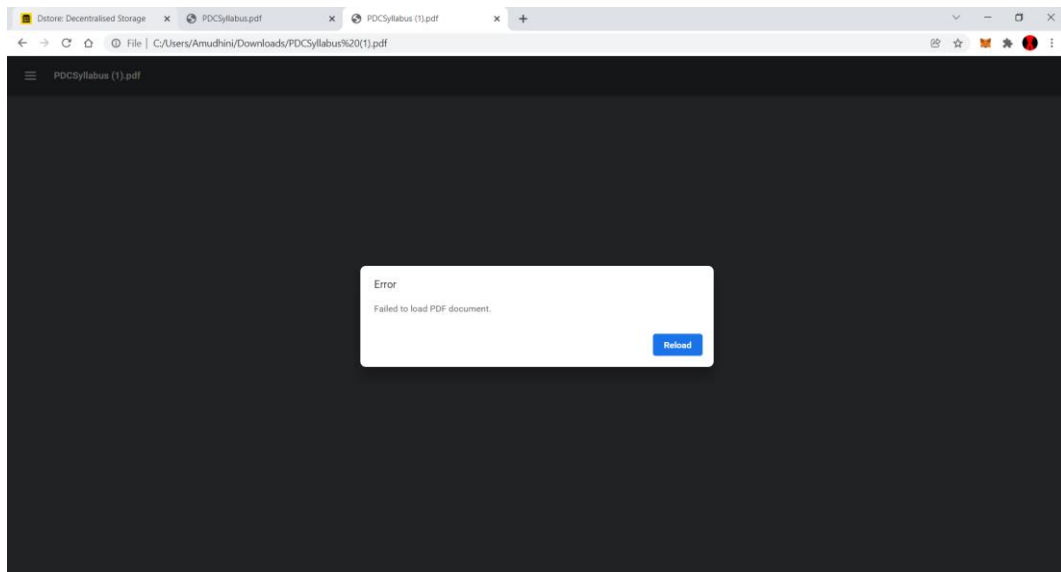
Downloading the file:

- When the user clicks download, a prompt pops up as shown asking for the password.
- If the user enters the correct password, the file will be decrypted properly and the file can be viewed as shown below.



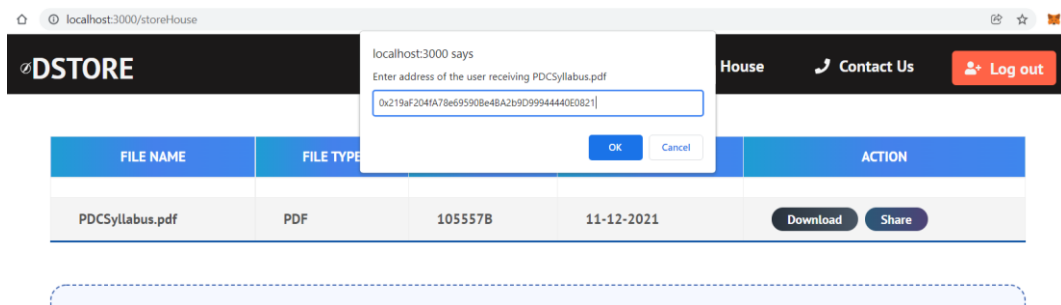
- If the user enters the a wrong password, the file is not successfully decrypted and an error occurs when trying to view the file.



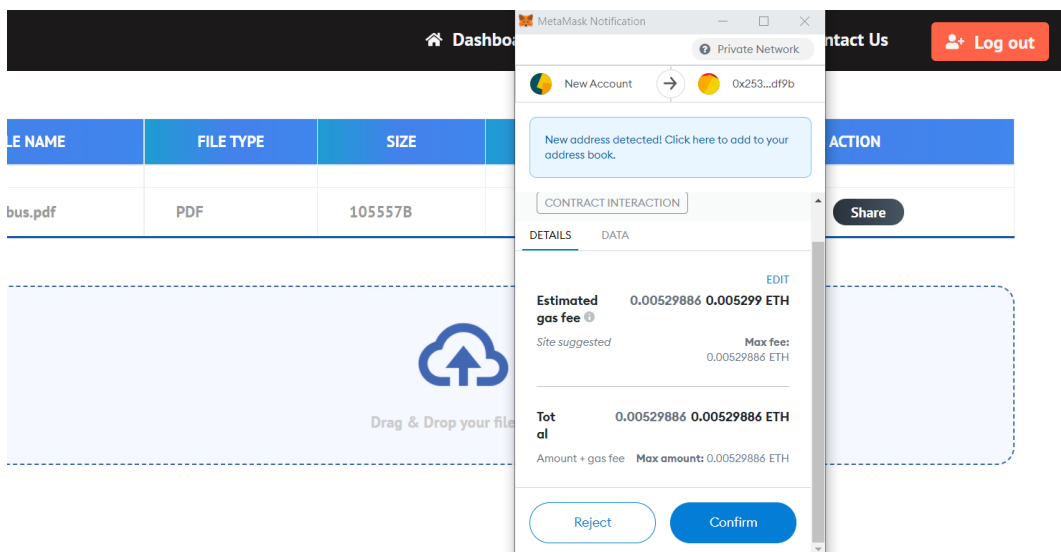


Share a file to another user:


- The user clicks the share button when he/she wants to share the file to another user in the application.
- A prompt pops up asking for the account address of the other user.



- Then the user pays the gas fee for successful transaction.



→ This is the dashboard of the other user before the transaction occurred.



Storage

4171.81KB

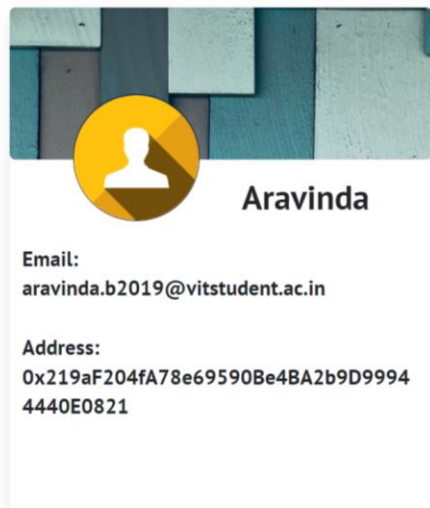
No of files stored

2

No of files shared by others

0

→ After the transaction, the number of shared files stored by the user has increased. There is also change in the storage and no of files stored.



Storage

4274.89KB

No of files stored

3

No of files shared by others

1

→ The file is also visible in the table along with the files uploaded by that user.

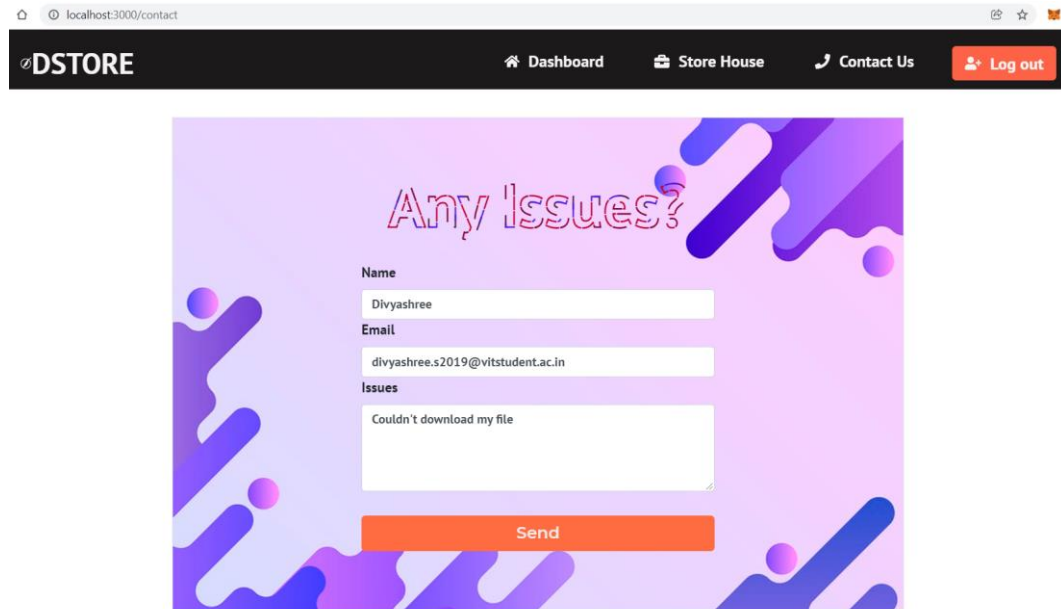
localhost:3000/storeHouse

DSTORE [Dashboard](#) [Store House](#) [Contact Us](#) [Log out](#)

FILE NAME	FILE TYPE	SIZE	UPLOAD DATE	ACTION
19BCE1190_1492_1689_PDC_Review-1.pdf	PDF	211696B	8-12-2021	Download Share
Aravinda.jpeg	JPEG	4060233B	10-12-2021	Download Share
PDCSyllabus.pdf	PDF	105557B	11-12-2021	Download Share

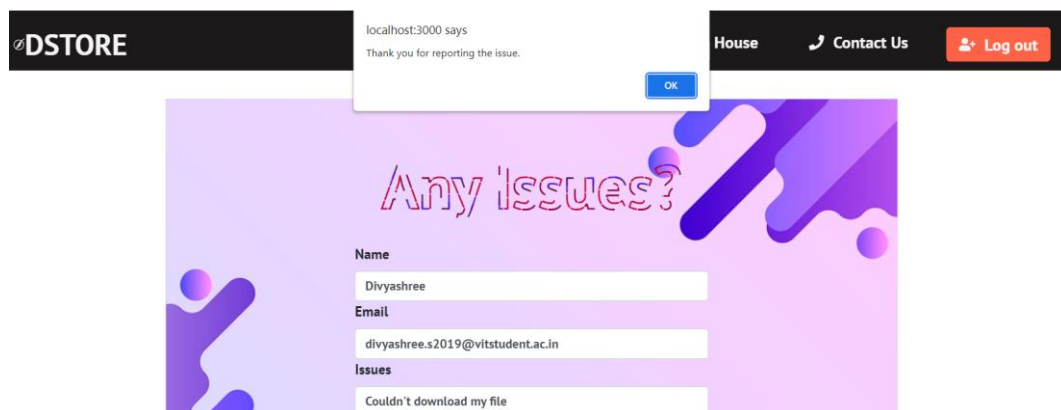
v. Contact Page:

- When the user has faces any issue in the application. He/she can report it in the contact us page.
- The user enters the issue faced and clicks send.



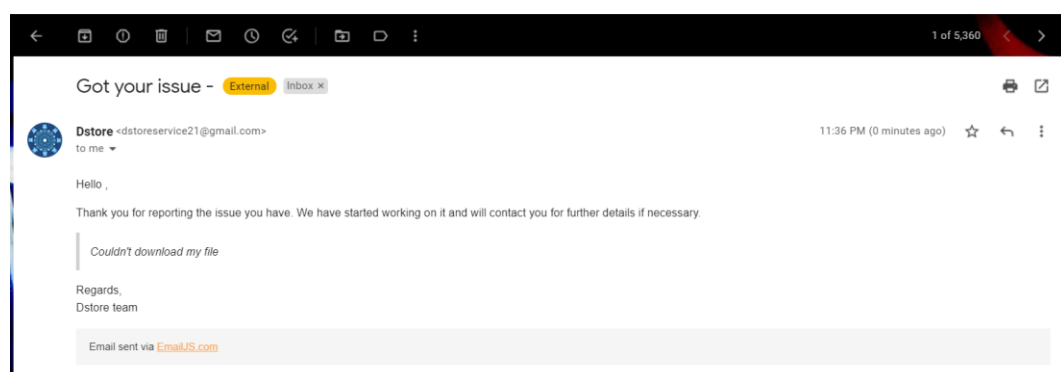
The screenshot shows a web browser at localhost:3000/contact. The page has a dark header with the DSTORE logo and navigation links: Dashboard, Store House, Contact Us, and a Log out button. The main content area has a purple and blue abstract background with the text "Any Issues?". Below this is a form with three input fields: "Name" (containing "Divyashree"), "Email" (containing "divyashree.s2019@vitstudent.ac.in"), and "Issues" (containing "Couldn't download my file"). A red "Send" button is at the bottom of the form.

- A message pops up acknowledging the report.



The screenshot shows the same contact form as before, but with a white confirmation message box overlaid. The message says "localhost:3000 says Thank you for reporting the issue." and has an "OK" button. The background of the page is the same purple and blue abstract design.

- A reply is also sent to the user's mail confirming that the application received the issue reported.



The screenshot shows an email interface with a message from "Dstore <dstore.service21@gmail.com>". The subject is "Got your issue - External". The email body says: "Hello , Thank you for reporting the issue you have . We have started working on it and will contact you for further details if necessary. Couldn't download my file Regards, Dstore team". The email was sent 11:36 PM (0 minutes ago). The interface shows a toolbar with various icons and a status bar at the bottom indicating "Email sent via EmailJS.com".

VII. SAMPLE CODE

Login contract: [Solidity]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.4.21;
3  pragma experimental ABIEncoderV2;
4
5  contract LoginInfo {
6      struct userStruct {
7          string username;
8          string email;
9      }
10
11     mapping (address => userStruct) public userMapping;
12
13     function registerUser(address _usraddr, string memory name,
14                           string memory email) public
15     {
16         userMapping[_usraddr]=userStruct(name,email);
17     }
18
19     function getUser(address _usraddr) public view returns
20         (userStruct memory _userStruct)
21     {
22         return userMapping[_usraddr];
23     }
24 }
```

File Storage contract: [Solidity]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.4.21;
3  pragma experimental ABIEncoderV2;
4
5  contract LoginInfo {
6      struct userStruct {
7          string username;
8          string email;
9      }
10
11     mapping (address => userStruct) public userMapping;
12
13     function registerUser(address _usraddr, string memory name,
14                           string memory email) public
15     {
16         userMapping[_usraddr]=userStruct(name,email);
17     }
18 }
```

```

17     function getUser(address _usraddr) public view returns
           (userStruct memory _userStruct) {
18         return userMapping[_usraddr];
19     }
20
21 }

```

Code snippet of User Login: [React]

```

1  handlesignIn = async () => {
2      const metamaskAddr = this.state.accounts[0];
3      console.log(metamaskAddr)
4      const contract = this.state.logincontract;
5
6      //Verify MetaMask account holder
7      const nonce = Math.floor(Math.random() * 10000);
8      var message = web3.utils.fromUtf8("One-time nonce: " + nonce);
9      var acc = await web3.eth.getAccounts()
10     var signature = await web3.eth.personal.sign(message, acc[0])
11
12     var hash = web3.utils.fromUtf8("One-time nonce: " + nonce)
13     var signing_address = await web3.eth.personal.ecRecover(hash,
                                                                    signature)
14
15     if (signing_address !== metamaskAddr.toLowerCase()) {
16         window.alert("Account verification failed. Try again.")
17         return;
18     }
19
20     const response = await contract.methods.getUser(
                                                                    metamaskAddr).call();
21
22     if (response['username'] === "" ||
23         response['email'] === "") {
24         window.alert("User does not exist. Please sign up");
25         return;
26     }
27
28     const username = response['username'];
29     const email = response['email'];
30     localStorage.setItem('UserName', username);
31     localStorage.setItem('Email', email);
32     localStorage.setItem('MetamaskAddr', metamaskAddr);
33
34     window.alert("Welcome back " + username);
35
36     window.location.reload();
37 }

```

Code snippet of User Signup: [React]

```
1  handlesignUp = async (event) => {
2
3    event.preventDefault();
4    const name = event.target[0].value;
5    const email = event.target[1].value;
6    const addr = event.target[2].value;
7
8    const metamaskAddr = this.state.accounts[0];
9
10   //Verify MetaMask account holder
11   const nonce = Math.floor(Math.random() * 10000);
12   var message = web3.utils.fromUtf8("One-time nonce: " + nonce);
13   var acc = await web3.eth.getAccounts()
14   var signature = await web3.eth.personal.sign(message, acc[0])
15
16   var hash = web3.utils.fromUtf8("One-time nonce: " + nonce)
17   var signing_address = await web3.eth.personal.ecRecover(hash,
18                                                                signature)
19
20   if (signing_address !== metamaskAddr.toLowerCase() ||
21       metamaskAddr !== addr) {
22     window.alert("Account verification failed. Try again.")
23     return;
24   }
25
26   const contract = this.state.logincontract;
27   const response = await contract.methods.getUser(
28                                                                metamaskAddr).call();
29
30   if (response['username'] !== "" ||
31       response['email'] !== "") {
32     window.alert("User already exists. Log in with your
33                                                                MetaMask.");
34     return;
35   }
36
37   await contract.methods.registerUser(metamaskAddr, name,
38                                                                email).send({ from: metamaskAddr });
39
40   window.alert("User successfully created");
41 }
```

Code snippet of File Upload [including encryption]: [React]

```
1  addNewFile = async (filename, filesize, uploaddate, ipfshash) => {
2    const contract = this.state.storagecontract;
3    const metamaskAddr = this.state.accounts[0];
4    await contract.methods.addFile(filename, filesize.toString(),
5      Math.floor(uploaddate), ipfshash).send({from: metamaskAddr})
6    window.location.reload();
7  }
8
9  onUploadSubmit = async () => {
10    var file, password, reader, wordArray, encrypted, ipfshash;
11    if (this.filesToUpload !== []) {
12      for (var i = 0; i < this.filesToUpload.length; i++) {
13        file = this.filesToUpload[i]
14        password = prompt("Enter password for " + file['name'])
15        if (password == null)
16          return
17
18        reader = new window.FileReader()
19        reader.readAsArrayBuffer(file)
20
21        reader.onload = () => {
22          // Convert: ArrayBuffer -> WordArray
23          wordArray = CryptoJS.lib.WordArray.create(reader.result);
24          // Encryption:I: WordArray -> O: -> Base64 encoded string
25          encrypted = CryptoJS.AES.encrypt(wordArray,
26            password).toString();
27
28          ipfs.files.add(Buffer(encrypted), (error, result) => {
29            if (error) {
30              console.error(error)
31              return
32            }
33            ipfshash = result[0].hash
34            console.log('ipfsHash', result[0].hash)
35
36            var date = (new Date()).getTime();
37            var dateUnixTimestamp = date / 1000;
38            console.log(file['name'], file['size'], date,
39              dateUnixTimestamp, ipfshash)
40            this.addNewFile(file['name'], file['size'],
41              dateUnixTimestamp, ipfshash)
42          })
43        }
44      }
45    }
46    else {
47      alert("Select files to upload")
48    }
49  }
```


Code snippet of File Download [including decryption]: [React]

```
1  handleDownload = async (fname, fhash) => {
2      let password = prompt("Enter password for " + fname)
3      if (password == null)
4          return
5      ipfs.files.get(fhash, function (err, files) {
6          files.forEach(function callback(file) {
7              console.log(file.path)
8              console.log("File content >> ",
                           file.content.toString('utf8'))
9              var decrypted = CryptoJS.AES.decrypt(file.content.toString(
                           'utf8'), password);
10
11              let wordArray = decrypted;
12              var arrayOfWords = wordArray.hasOwnProperty("words") ?
                           wordArray.words : [];
13              var length = wordArray.hasOwnProperty("sigBytes") ?
                           wordArray.sigBytes : arrayOfWords.length * 4;
14
15              var uInt8Array = new Uint8Array(length), index = 0, word, i;
16
17              for (i = 0; i < length; i++) {
18                  word = arrayOfWords[i];
19                  uInt8Array[index++] = word >> 24;
20                  uInt8Array[index++] = (word >> 16) & 0xff;
21                  uInt8Array[index++] = (word >> 8) & 0xff;
22                  uInt8Array[index++] = word & 0xff;
23              }
24
25              let typedArray = uInt8Array;
26
27              var downloadBlob = function (data, fileName, mimeType) {
28                  var blob, url;
29                  blob = new Blob([data], {
30                      type: mimeType
31                  });
32                  url = window.URL.createObjectURL(blob);
33                  downloadURL(url, fileName);
34                  setTimeout(function () {
35                      return window.URL.revokeObjectURL(url);
36                  }, 1000);
37              };
38
39              var downloadURL = function (data, fileName) {
40                  var a;
41                  a = document.createElement('a');
42                  a.href = data;
43                  a.download = fileName;
44                  document.body.appendChild(a);
45                  a.style = 'display: none';
```

```
46         a.click();
47         a.remove();
48     };
49     downloadBlob(typedArray, `${fname}`);
50
51     })
52 })
53
54 }
```

Code snippet of File Share:

```
1  handleShare = async (file) => {
2      var address = prompt("Enter address of the user receiving " +
                           file[0])
3
4      if (address == null)
5          return
6
7      try {
8          const contract = this.state.storagecontract;
9          const metamaskAddr = this.state.accounts[0];
10
11          var date = (new Date()).getTime();
12          var dateUnixTimestamp = date / 1000;
13
14          await contract.methods.shareFile(address, file[0], file[1],
15              Math.floor(dateUnixTimestamp), file[2]).send({
16              from: metamaskAddr })
17
18          alert("File sent to " + address)
19      }
20      catch (error) {
21          alert("Invalid metamask address" + error)
22          return;
23      }
24
25  }
26 }
```

VIII. CONCLUSION

The proposed system enhances the security of data by encrypting and distributing the data across multiple peers in the system. The implemented system encrypts data using the AES 256 bit encryption technique, assuring the privacy of the user's data. The IPFS protocol is then used to distribute and store encrypted data among network peers. The system not only addresses the privacy and security issues associated with centralized cloud storage, but also provides a platform for peers to rent out underused storage and earn cryptocurrency in exchange, increasing storage resource usage.

In the future, an adaptive scheduling algorithm could be implemented so that files can be viewed several times by the user rather than being accessible just once. This will make it easier for the user to access commonly used files whenever they are needed. A credit system can also be introduced, with each peer receiving a default 100 credit depending on their system uptime, and multiple successfully served file access requests having their credits subtracted or added. Data storage preference will be given to peers with greater credits.

IX. REFERENCES

- [1] Chernet, Haimanot Fiseha and Sunil Kumar Jilleli. "A Next-Generation Smart Contract and Decentralized blockchain Platform: A case study on Ethiopia." (2020).
- [2] Li, Dagang, Rong Du, Yue Fu and Man Ho Au. "Meta-Key: A Secure Data-Sharing Protocol Under Blockchain-Based Decentralized Storage Architecture." *IEEE Networking Letters* 1 (2019): 30-33.
- [3] Zhe, Diao, Wang Qinghong, Su Naizheng and Zhang Yuhan. "Study on Data Security Policy Based on Cloud Storage." 2017 IEEE 3rd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS) (2017): 145-149.
- [4] Zhu, Yan, Chunli Lv, Zichuan Zeng, Jingfu Wang and Bei Pei. "Blockchain-based Decentralized Storage Scheme." *Journal of Physics: Conference Series* (2019): n. pag.
- [5] Zyskind, Guy, Oz Nathan and Alex 'Sandy' Pentland. "Decentralizing Privacy: Using Blockchain to Protect Personal Data." 2015 IEEE Security and Privacy Workshops (2015): 180-184.