CS4037D CLOUD COMPUTING
Result Analysis

Submitted by:
Abin K Paul - B170217CS
Robin Ainikkal - B170169CS
Rahul Ram KK - B170776CS

Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

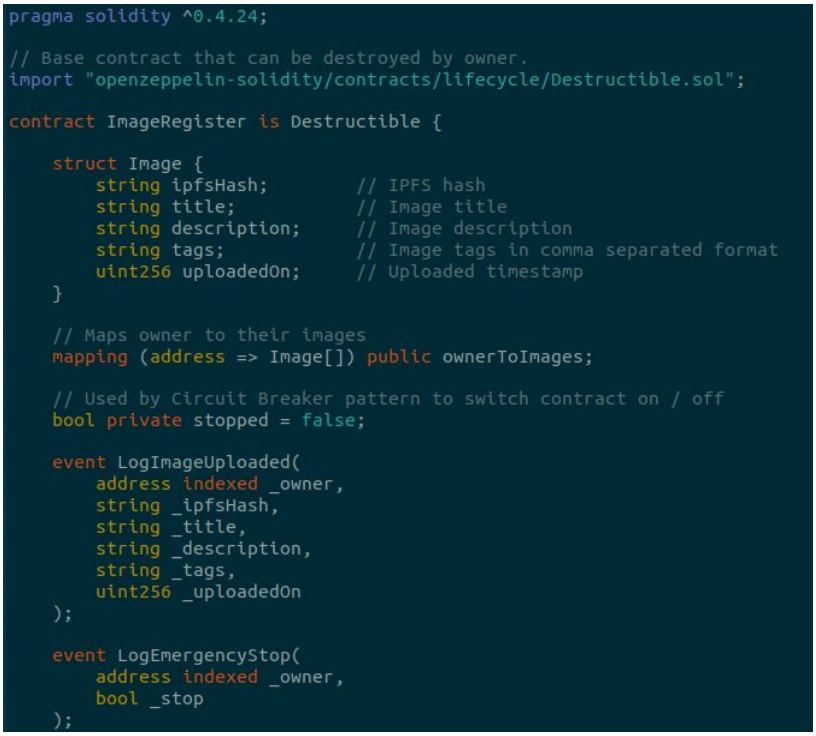
Content

IPFS storage Based Government Documents Portal Using Blockchain

1. Components of the Project

1.1 Smart Contracts

Solidity is used to write smart contracts. We use solidity version 0.4.24.
We have two contracts for the project:
ImageRegister Contract :-
This contract defines a structure to store data about each document image uploaded. It holds the image title, description, uploaded timestamp and optional tags. It also holds the ipfsHash which is used to locate the document on IPFS. Then we have a mapping to store all the documents. We define an event to log the data about the image.



```
pragma solidity ^0.4.24;

// Base contract that can be destroyed by owner.
import "openzeppelin-solidity/contracts/lifecycle/Destructible.sol";

contract ImageRegister is Destructible {

    struct Image {
        string ipfsHash;        // IPFS hash
        string title;           // Image title
        string description;     // Image description
        string tags;            // Image tags in comma separated format
        uint256 uploadedOn;     // Uploaded timestamp
    }

    // Maps owner to their images
    mapping (address => Image[]) public ownerToImages;

    // Used by Circuit Breaker pattern to switch contract on / off
    bool private stopped = false;

    event LogImageUploaded(
        address indexed _owner,
        string _ipfsHash,
        string _title,
        string _description,
        string _tags,
        uint256 _uploadedOn
    );

    event LogEmergencyStop(
        address indexed _owner,
        bool _stop
    );
```

```
function getImageCount(address _owner)
    public view
    stopInEmergency
    returns (uint256)
{
    require(_owner != 0x0);
    return ownerToImages[_owner].length;
}

function getImage(address _owner, uint8 _index)
    public stopInEmergency view returns (
    string _ipfsHash,
    string _title,
    string _description,
    string _tags,
    uint256 _uploadedOn
) {
    Image storage image = ownerToImages[_owner][_index];

    return (
        image.ipfsHash,
        image.title,
        image.description,
        image.tags,
        image.uploadedOn
    );
}
```

We then define a function to upload document images. It creates a new image object, assigns an owner and stores it on the blockchain and logs the result.

We also define a function to get the number of documents saved and also retrieve uploaded document images by passing in the address of the owner.

```
function uploadImage(
    string _ipfsHash,
    string _title,
    string _description,
    string _tags
) public stopInEmergency returns (bool _success) {
    uint256 uploadedOn = now;
    Image memory image = Image(
        _ipfsHash,
        _title,
        _description,
        _tags,
        uploadedOn
    );

    ownerToImages[msg.sender].push(image);

    emit LogImageUploaded(
        msg.sender,
        _ipfsHash,
        _title,
        _description,
        _tags,
        uploadedOn
    );

    _success = true;
}
```

Migrations Contract :-
This contract keeps track of which migrations were done on the current network

1.2 Ganache

Ganache is used to quickly set up Ethereum blockchain.
When we set up Ganache, ten accounts with hundred ether each. While on the local network, we can use these to save documents to the blockchain.

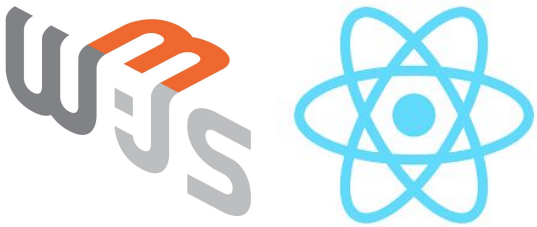
```
Ganache CLI v6.9.1 (ganache-core: 2.10.2)

Available Accounts
==================
(0) 0xA1B9dd575FF681B8D1A65DEa2CCFEe116747f31e (100 ETH)
(1) 0xbc2CCa7667c2860e388058715f88cf8683E8d335 (100 ETH)
(2) 0x87DF1F6dc1C4D89B2D40317A81bfd2650f280CA4 (100 ETH)
(3) 0x49420BfaE4650A548dD7B3Ae3aBC1cA46b102f6c (100 ETH)
(4) 0x2256F28788451e554aB329929bEc3806246eCC0a (100 ETH)
(5) 0x16FEe027F312B6c501e744f3C530DA14608d7097 (100 ETH)
(6) 0xbE14F1E0e843B1827FCd54180f6aF5bE9751a5Dc (100 ETH)
(7) 0x105401C44804f5dAb89ac78D2fd8A6110aA46BA0 (100 ETH)
(8) 0x95e8Ecb961D39ae86Cd7341Dd9b2ae20f523624a (100 ETH)
(9) 0x88D3553BaBFD37a3C38C8e79beDC8E15a9B0a19A (100 ETH)

Private Keys
==================
(0) 0xe247561c1bbf3ec007b4dbacdcff0910bafc53deee2ed681dd7d26e9d4909c4a
(1) 0x00cf32c4091a8f330ee2b8de85510d56f2c969e14caa3213010b8985014a1146
(2) 0xf4f34b11b8896f0d5ac4bfa10475bd268f1f430fb4ba36dbc2a612ebbf33efb3
(3) 0x480dd33885b1b119340c408a99413e18c271165f0e94ca10e3213f82faed9b2a
(4) 0x9b5eddf9c6341a399e8496221bf5bd37aa22ed21dac23da80afd7980d8a8f589
(5) 0xda8f6a8e7424b5f350cdacbec79aaaffd66a767f2e36d92488f381bb96ceea1e
(6) 0x0544be58b27735c14256690cc093d8a891fd9b109074cc073e122377a25d4cff
(7) 0x237c2ea6e1163c5cf6aca06811ab4ff5b2f7d6da16e10fe09d37ad552df757c5
(8) 0x0ef57ade81976e1743454dfd4a73da9ab0a7dc1b2d7e83fee08d74496d13b7a2
(9) 0x0cd9939fe6fa7bc1c647427fa367f4a86905447134a0c900af11fbcde32aea60

HD Wallet
==================
Mnemonic:      fold pull month coast agent angry domain arena wide select leisure nest
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Call Gas Limit
==================
9007199254740991

Listening on 127.0.0.1:8545
eth_blockNumber
eth_blockNumber
```

1.3 Truffle

Truffle is used to compiling the smart contracts. We first migrate changes made to the contract and then compile them. After compiling truffle can deploy the contracts to any blockchain network we specify. Here we compile and deploy to the Ganache Blockchain Network

1.4 Web3 and React

We make use of React to organize the front end of the website. Web3 provides APIs to interact directly with the smart contracts on the Blockchain network using Javascript.



```
// Add image to IPFS
ipfs.files.add(buffer, async (error, result) => {
  if (error) {
    console.log('ERR', error)
    dispatch({
      type: SET_ERROR,
      payload: {
        error,
      },
    })
  } else {
    const ipfsHash = result[0].hash // base58 encoded multihash
    ipfs.files.get(ipfsHash, (error, files) => {
      console.log(files)
```

```
const web3State = getState().web3
const contractInstance = web3State.contractInstance
try {
  // Success, upload IPFS and metadata to the blockchain
  const txReceipt = await contractInstance.uploadImage(
    ipfsHash,
    title,
    description,
    tags,
    {
      from: web3State.account,
    }
  )

  console.log('uploadImage tx receipt', txReceipt)
```

Fig : Core logic for uploading the document image to IPFS and then storing hash and data on the blockchain.

```
// Get all images
export const getImages = () => async (dispatch, getState) => {
  dispatch({ type: GET_IMAGES })

  const web3State = getState().web3

  // Retrieve image state from local storage
  const localData = localStorage.getItem(web3State.account)
  const localImages = localData ? JSON.parse(localData) : []
  const imagesByIndex = keyBy(localImages, 'index')

  const images = []
  try {
    const count = await web3State.contractInstance.getImageCount.call(
      web3State.account,
      {
        from: web3State.account,
      }
    )
    const imageCount = count.toNumber()
   for (let index = 0; index < imageCount; index++) {
     const imageResult = await web3State.contractInstance.getImage.call(
       web3State.account,
       index,
       {
         from: web3State.account,
       }
     )

     // Image for UI
     const image = {
       ...imagesByIndex[index],
       index,
       ipfsHash: imageResult[0],
       title: imageResult[1],
       description: imageResult[2],
       tags: imageResult[3],
       uploadedOn: convertTimestampToString(imageResult[4]),
     }
     images.push(image)
   }
```

```
  // Save image state to local storage
  localStorage.setItem(web3State.account, JSON.stringify(images))

  dispatch({ type: GET_IMAGES_SUCCESS, payload: images })
} catch (error) {
  console.log('error', error)
  dispatch({ type: SET_ERROR, payload: error })
}
```

Fig : Core logic to retrieve information from blockchain and subsequently the document image from IPFS

2. Deployment

2.1 Configuration of Deployment

```
const HDWalletProvider = require('truffle-hdwallet-provider')

module.exports = {
  networks: {
    development: {
      host: 'localhost',
      port: 8545,
      network_id: '*', // Match any network id
    },
    ropsten: {
      provider: () =>
        new HDWalletProvider(
          process.env.MNENOMIC,
          'https://ropsten.infura.io/' + process.env.INFURA_API_KEY
        ),
      network_id: 3,
    },
  },
  compilers: {
    solc: {
      version: '0.4.24',
    },
  },
}
```

2.2 Compiling and Deploying Smart Contracts using Truffle

```
Compiling your contracts...
===========================
> Compiling ./contracts/ImageRegister.sol
> Compiling ./contracts/Migrations.sol
> Compiling openzeppelin-solidity/contracts/lifecycle/Destructible.sol
> Compiling openzeppelin-solidity/contracts/ownership/Ownable.sol
> Artifacts written to /home/drbinu/Desktop/Projects-Works/ipfs-image-dapp/build/contracts
> Compiled successfully using:
   - solc: 0.4.24+commit.e67f0147.Emscripten.clang



Starting migrations...
======================
> Network name:    'development'
> Network id:      1595232888877
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xda4d9e01153e76df2f7eb605f8c6ca4778bb63c5e5033eedb9fd61dac28480a1
   > Blocks: 0            Seconds: 0
   > contract address:    0x3eCA8418AbeFF9F47dC4EF6BeB0268a7f19C44f7
   > block number:        1
   > block timestamp:     1595234521
   > account:             0xA1B9dd575FF681B8D1A65DEa2CCFEe116747f31e
   > balance:             99.99522812
   > gas used:            238594 (0x3a402)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00477188 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00477188 ETH

2_deploy_contracts.js
=====================

   Deploying 'ImageRegister'
   -------------------------
   > transaction hash:    0x18d59b4fd040e1fc8612714a7d9da3be94c7d5cc1bbd7d6245d050432b7464ca
   > Blocks: 0            Seconds: 0
   > contract address:    0x4109A020252a5Ec35b143c011Ef8F04bfdaeD06b
   > block number:        3
   > block timestamp:     1595234521
   > account:             0xA1B9dd575FF681B8D1A65DEa2CCFEe116747f31e
   > balance:             99.96645962
   > gas used:            1396077 (0x154d6d)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02792154 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.02792154 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.03269342 ETH
```

2.3 Connecting Metamask to our wallet

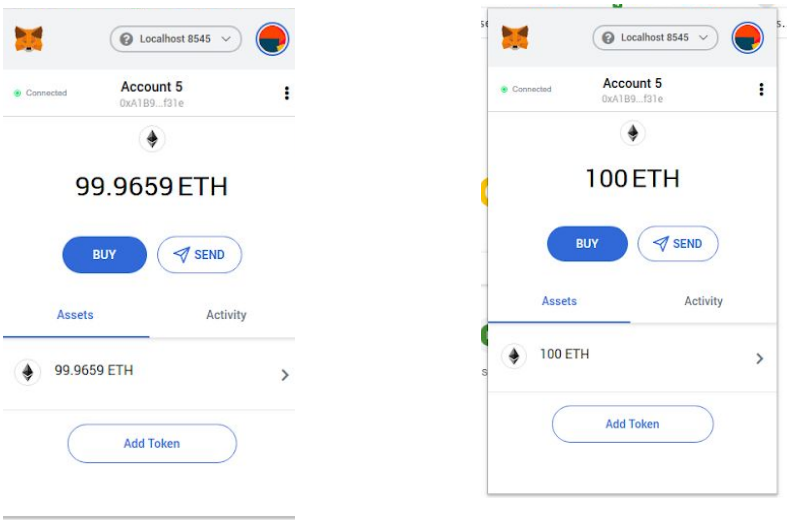


Fig: Wallet balance before and after deployment

We enable Metamask extension in the browser. Metamask is used to connect to our ethereum accounts on Ganache. So using Metamask we first import an account that has sufficient ether to do the image upload.

2.4 Starting the React WebApp

The react app is served using node packet manager. The landing page consists of the already stored documents and an option to go to the upload page.

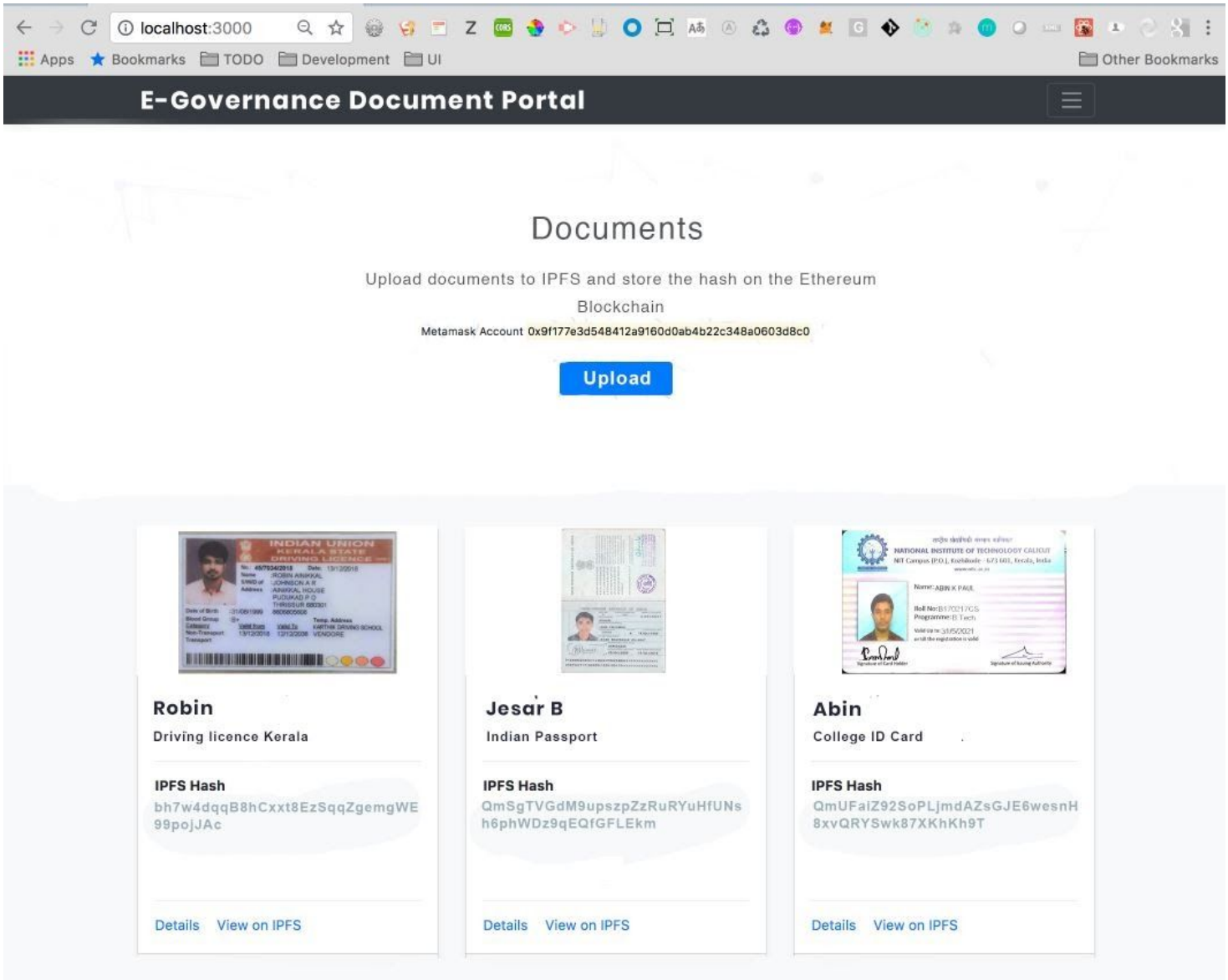


Fig: Landing page

On the upload page we can provide the document image and also the basic details like name and type of document. Metamask will prompt for confirmation when upload is confirmed.

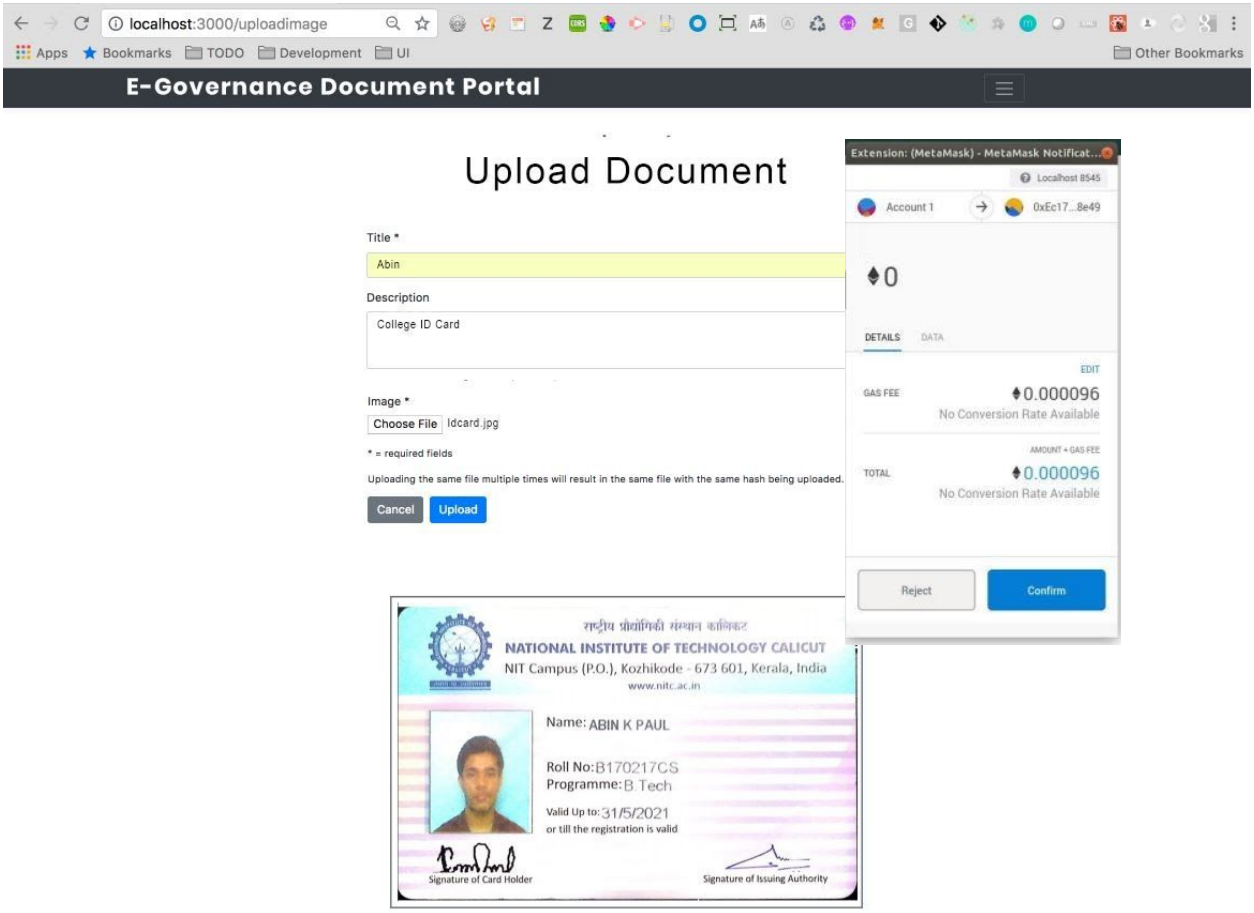


Fig: Upload page

The individual document details can also be viewed by clicking the Details button for each entry on the landing page. To view the document on IPFS click the ‘View on IPFS’ and will be redirected to the ipfs link.

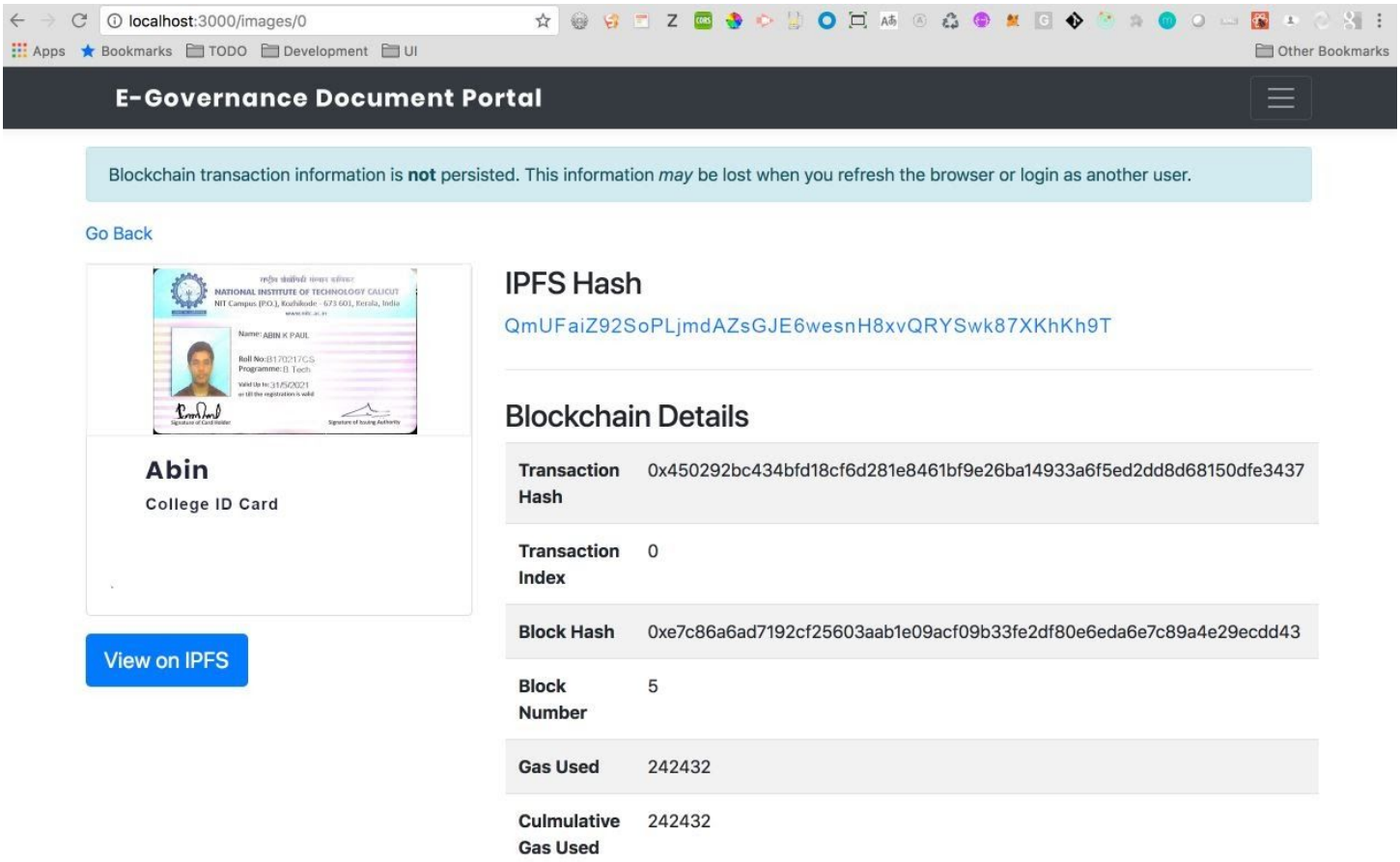


Fig: View Details Page

3. Conclusion

Based on the working of the developed system we can make a few conclusions. They are as follows:

- Fake documents can be traced back to the account which added it. Since all records on blockchain are public, accounts which try to add fake documents can be traced back and handled appropriately.
- Prevents fraud. Any document can be easily cross verified with the document on the blockchain to establish authenticity
- Third parties should be unable to tamper official documents.. Due to the immutability of the blockchain system, tampering of details and documents is not possible.
- No single point of failure. Due to the distributed nature of IPFS multiple copies are persisted across the network.
- Ease of access. Document stored on IPFS is easily accessed from the closest peer. Reliance on a single server is reduced