

**DECENTRALIZED SOCIAL MEDIA APPLICATION
WITH REWARD SYSTEM**

PROJECT REPORT

Submitted by

Srivatsav R

Sachin Raghul T

Sanjeev K M

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2022

**ANNA UNIVERSITY: CHENNAI 600
025**

BONAFIDE CERTIFICATE

Certified that this project report **“DECENTRALIZED SOCIAL MEDIA APPLICATION WITH REWARD SYSTEM”** is the bonafide work of **“Srivatsav R, Sachin Raghul T, Sanjeev K M”** who carried out the project work under my supervision.

SIGNATURE

SIGNATURE

HEAD OF THE DEPARTMENT

SUPERVISOR

TABLE OF CONTENTS

S No.	TITLE	PAGE NO
	ABSTRACT	4
1.	INTRODUCTION	5
2.1	LITERATURE SURVEY	6
2.2	SUMMARY	9
3	PROBLEM STATEMENT	13
4.	OBJECTIVE	13
5.	SYSTEM ARCHITECTURE	14
6.	LIST OF MODULES	15
7.1	EXPLANATION OF MODULES: PROCESS FOR REGISTERING	15
7.2	EXPLANATION OF MODULES: PROCESS FOR CREATING POST	16
7.3	EXPLANATION OF MODULES: REWARD SYSTEM	16
8.	IMPLEMETATION	17
9.	CONCLUSION	32
10.	REFERENCES	33

ABSTRACT

The key to this decentralized paradigm is not merely security, which is not too hard with public key cryptography, but user-friendly security, which lets us have the conveniences we're used to in centralized systems, but keeps the network secure and open to anyone interfacing with it in whatever way they please.

In this project, we propose **Ethegram**, a social media platform developed by using the state-of-the-art **blockchain** technology and introduce a social networking decentralized application. The application is based on **Ethereum blockchain** platform for data records and **IPFS** for distributed data storage service. The application would reward users for quality content they contribute on the platform.

IPFS - Inter Planetary File System

We achieve these features, including confidentiality, metadata hiding, profiles, friend networks, instant messaging, groups, and much more through a carefully constructed profile file tree distributed peer to peer over IPFS. It can be easily updated, distributed via deltas rather than bulk transfers, and hosted without being able to glean any information about that user. Offline or network-partitioned use is natural for our system, and it will handle such network difficulties gracefully—spreading data to what peers it can reach and recovering effortlessly once the partition is resolved. Most importantly of all, it lays the foundation for a secure system that people may actually want to use.

Reward system

When the user creates a post, other users of the application can view and like the post if they like the content. Each like generates a single token in the application account of the creator. These tokens can be redeemed into Ethers from the platform itself. The funds are directly transferred to the account address from the contract balance. Each token when redeemed gives 0.0001 Ether. Thus, the cost of creating a new post is recovered when the user receives 10 likes on his post. More likes would ensure that the one-time registration cost is also recovered. Content creators are rewarded only based on how much their content is liked by other users of the application.

INTRODUCTION

Ethegram An Ethereum and IPFS-based Decentralized Social Network with Reward System

PROPOSAL:

Social media applications like facebook provides decent service at the extremely high cost of constantly disregarding user privacy, common decency, and laws around the world, and yet they still have billions of users as they are the monopoly. Monopolies breed complacency, low quality, and high costs. And Facebook has a monopoly over your friend network. We can't leave because everyone is on Facebook. Switching to a new platform means every single one of your friends has to make a new account, download a new app, and you have to rebuild that whole network—if you can convince them at all. It's the same story for any platform (though most don't charge such a high price), and it's unavoidable.

We believe the solution is a decentralized social network which is encrypted at rest. When the user has the key to decrypt and modify their own data, they have complete control, and can grant and revoke control from third parties. Everyone's data is just 'out there', many copies floating around in encrypted blobs that anyone can host or download but only friends can decrypt. Decentralization also provides robustness against censorship, internet outages, and would-be social monopolies.

DOMAIN:

BLOCKCHAIN, ETHEREUM, IPFS

DELIVERABLES:

In this project, we propose **Ethegram**, a social media platform developed by using the state-of-the-art **blockchain** technology and introduce a social networking decentralized application. The application is based on **Ethereum blockchain** platform for data records and **IPFS** for distributed data storage service. The application would reward users for quality content they contribute on the platform.

LITERATURE SURVEY

#	Author, Publication Year, Title	Proposed work	Algorithm Used	Advantages	Limitations
1.	Ningyuan Chen, David Siu-Yeung Cho, A Blockchain based Autonomous Decentralized Online Social Network, 2021	Online Social Network (OSN) using Decentralized Autonomous Organization (DAO)	Decentralized Autonomous Organization (DAO) and IPFS based algorithm	DAO is developed for user autonomy, users can self-manage the OSN in a democratic way	The simulate plan doesn't motivate users to create more high-quality content and pay their effort in the autonomy part
2.	Barbara Guidi An Overview of Blockchain Online Social Media from the Technical Point of View 2021	Blockchain technology to decentralize control on social data and users maintain control of their social content.	Delegated Proof of Stake (DPoS)	Users gain full control of their content and are rewarded in order to encourage engagement, Participation.	Ethereum, can help to improve its application in a social scenario than Steem and hive
3.	Q. Xu, K. M. M. Aung, Y. Zhu, and K. L. Yong, New Advances in the Internet of Things. Springer, 2018, pp. 119–138.	A blockchain-based storage system for data analytics in the internet of things	OSD-based smart contract (OSC)	Secured blockchain based storage	Hard to implement it's prototype
4.	IPFS. [Online]. Available: https://ipfs.io/ (2018)	Interplanetary file system for handling file uploads	Peer 2 peer algorithm	Interplanetary file system for handling file uploads is best way	IPFS consumes a lot of bandwidth which is not appreciated by metered internet users.

5.	A. Tar. (July 26 2018) Smart contracts, explained. [Online]. Available: https://cointelegraph.com/explained/smart-contracts-explained .	Explanation for the programs stored on a blockchain that run when predetermined conditions are met.	Ethereum smart contracts	automatically controls the transfer of digital assets between the parties under certain conditions	Difficult to change. Changing smart contract processes is almost impossible, any error in the code can be time-consuming and expensive to correct.
6.	Antorweep Chakravorty and Chunming Rong, Ushare, Conference Paper – January 2017.	User controlled social media based on blockchain	Personal Certificate Authority (PCA) DECENT algorithm	support offsite encryption of data and mechanisms to share them through the blockchain.	a token value reaches zero, it reverts to the previous state and does not allow any future share of that data item.
7.	Q. Xu, C. Jin, M. F. B. M. Rasid, B. Veeravalli, and K. M. M. Aung, Multimedia Tools and Applications, pp. 1–26, 2017	Blockchain-based decentralized content trust for docker images	DTC algorithm	protection against a single point of failure removes the need of a central authority to manage trust	Limited Protection against Server Compromise Potential DoS Attack

8.	Buterin, Ethereum White Paper. 2014, p.23.	A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM	Ethereum smart contracts	Backup, speed, safety, accuracy is obtained on using Ethereum smart contracts.	Possibility of loopholes and vague terms
9.	M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. In 2014 IEEE Symposium on Security and Privacy, pages 443–458. IEEE, 2014	Secure multiparty computations on bit coin	Bitcoin cryptocurrency wallet	Secure and no charge backs (lower transaction fees)	The value of Bitcoins is constantly fluctuating according to demand which would not please the user that the reward we are offering
10.	Datta, A.; Buchegger, S.; Vu, L.H.; Strufe, T.; Rzedca, K. In Handbook of Social Network Technologies and Applications; Springer	Decentralized Online Social Networks	distributed ledger technology, cryptography and P2P (peer-to-peer)	develop a distributed algorithm to detect the dynamic community	By analyzing the dynamic results, the network is very shattered and not even close to the static view.

SUMMARY OF LITERATURE SURVEY:

1) A Blockchain based Autonomous Decentralized Online Social Network

Online social networks (OSN) are becoming more important in people's daily life, however, all popular OSNs are centralized, and this raises a series of security, privacy and management issues. A decentralized architecture based on blockchain technology provides the ability to solve above issues. In this paper, an OSN service is developed based on blockchain technology in order to make it operate decentralized. Large volume of data normally required low-security requirements can be stored in Interplanetary Filesystem (IPFS) to make data decentralized. A decentralized autonomous organization is developed for user autonomy, users can self-manage the OSN in a democratic way.

2) An Overview of Blockchain Online Social Media from the Technical Point of View

Social media is becoming one of the dominant ways to communicate. Before social media, people were extremely limited in their means to interact with others, and they were limited largely to the people that they knew in person. However, this impact on people in real life has damaged privacy. Alternative solutions have been proposed in order to overcome current social media issues. In this direction, blockchain is one of the most promising, and several blockchain-based social media have been proposed. In this paper, we analyze blockchain online social media from the technical point of view in order to understand the current trend of social DApps and to describe which characteristics are important in a blockchain-based social media scenario.

3) New Advances in the Internet of Things.

The Internet of Things, or IoT, is the name that the IT folks have given to the now billions of physical devices throughout the world that are connected to the internet. These devices not only collect data, but they share it as well. Through the proliferation of wireless networks as well as cheap processors, it's now possible to turn a multitude of physical objects into an IoT device. These things can range from a smart thermostat in your home that you control with your smartphone, to a sophisticated driverless car that's filled with hundreds of sensors collecting and transmitting data back to make sure it is operating efficiently.

4) IPFS

The InterPlanetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices. IPFS allows users to host and receive content in a manner similar to BitTorrent. As opposed to a centrally located server, IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing. Any user in the network can serve a file by its content address, and other peers in the network can find and request that content from any node who has it using a distributed hash table (DHT).

5) Smart contracts

A smart contract is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. The objectives of smart contracts are the reduction of need in trusted intermediators, arbitrations and enforcement costs, fraud losses, as well as the reduction of malicious and accidental exceptions.

6) Ushare

This work provides a systematic literature review of blockchain-based applications across multiple domains. The aim is to investigate the current state of blockchain technology and its applications and to highlight how specific characteristics of this disruptive technology can revolutionise “business-as-usual” practices. To this end, the theoretical underpinnings of numerous research papers published in high ranked scientific journals during the last decade, along with several reports from grey literature as a means of streamlining our assessment and capturing the continuously expanding blockchain domain, are included in this review. Based on a structured, systematic review and thematic content analysis of the discovered literature, we present a comprehensive classification of blockchain-enabled applications across diverse sectors such as supply chain, business, healthcare, IoT, privacy, and data management, and we establish key themes, trends and emerging areas for research. We also point to the shortcomings identified in the relevant literature, particularly limitations the blockchain technology presents and how these limitations spawn across different sectors and industries. Building on these findings, we identify

various research gaps and future exploratory directions that are anticipated to be of significant value both for academics and practitioners.

7) Multimedia Tools and Applications

Multimedia Tools and Applications welcomes submissions for the new Tracks on Medical Applications of Multimedia, Biometrics and HCI, Digital Games and VR/AR, and Multimedia and Education. Multimedia Tools and Applications publishes original research articles on multimedia development and system support tools as well as case studies of multimedia applications. It also features experimental and survey articles. The journal is intended for academics, practitioners, scientists and engineers who are involved in multimedia system research, design and applications.

8) Ethereum White Paper

Being the first example of a digital asset, which simultaneously has no backing or "intrinsic value" and no centralized issuer or controller. However, another, arguably more important, part of the Bitcoin experiment is the underlying blockchain technology as a tool of distributed consensus, and attention is rapidly starting to shift to this other aspect of Bitcoin. Commonly cited alternative applications of blockchain technology include using on-blockchain digital assets to represent custom currencies and financial instruments ("colored coins"), the ownership of an underlying physical device ("smart property"), non-fungible assets such as domain names ("Namecoin"), as well as more complex applications involving having digital assets being directly controlled by a piece of code implementing arbitrary rules ("smart contracts") or even blockchain-based "decentralized autonomous organizations" (DAOs).

9) Symposium on Security and Privacy

The blockchain has fueled one of the most enthusiastic bursts of activity in applied cryptography in years, but outstanding problems in security and privacy research must be solved for blockchain technologies to go beyond the hype and reach their full potential. At the first IEEE Privacy and Security on the Blockchain Workshop (IEEE S&B), we presented peer-reviewed papers bringing together academia and industry to analyze problems ranging from deploying newer cryptographic primitives on Bitcoin to enabling usecases like privacy-preserving file storage. We overview not only the larger problems the workshop has set out to tackle, but also outstanding unsolved issues that will require further cooperation between academia and the blockchain community.

10) In Handbook of Social Network Technologies and Applications

Provides current and future trends in creating intelligent social networks, and the main players and their social networks applications. Presents web-mining techniques, visualization techniques, social networks and Semantic Web, and many other topics. Includes contributions from world experts in the field of social networks from both academia and private industry. Presents standards for social networks, case studies, and a variety of applications.

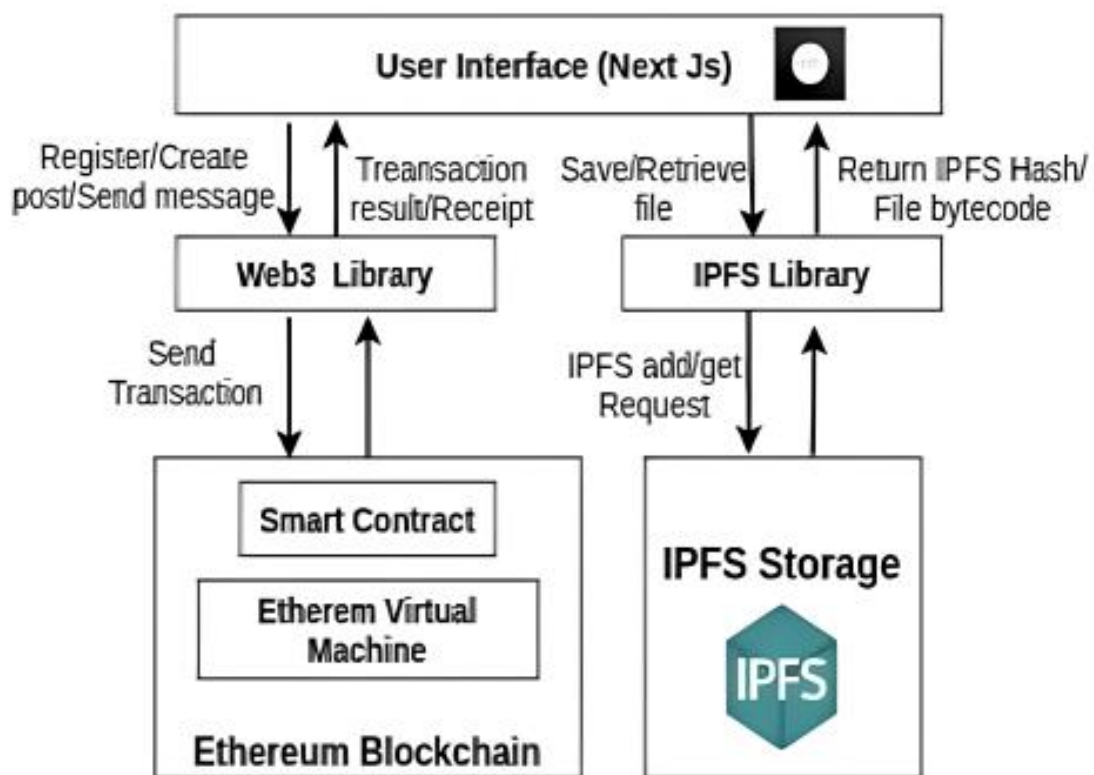
PROBLEM STATEMENT

As we are aware, Facebook provides decent service at the extremely high cost of constantly disregarding user privacy, common decency, and laws around the world, and yet they still have billions of users as they are the monopoly. Monopolies breed complacency, low quality, and high costs. And Facebook has a monopoly over your friend network. We can't leave because everyone is on Facebook. Switching to a new platform means every single one of your friends has to make a new account, download a new app, and you have to rebuild that whole network—if you can convince them at all. It's the same story for any platform (though most don't charge such a high price), and it's unavoidable. Hence, we believe the solution is a decentralized social network which is encrypted.

OBJECTIVE

Online social networks (OSN) are becoming more important in people's daily life, however, all popular OSNs are centralized, and this raises a series of security, privacy and management issues. A decentralized architecture based on blockchain technology provides the ability to solve above issues. In this paper, an OSN service is developed based on blockchain technology in order to make it operate decentralized. Large volume of data normally required low-security requirements can be stored in Interplanetary Filesystem (IPFS) to make data decentralized. A decentralized autonomous organization is developed for user autonomy, users can self-manage the OSN in a democratic way.

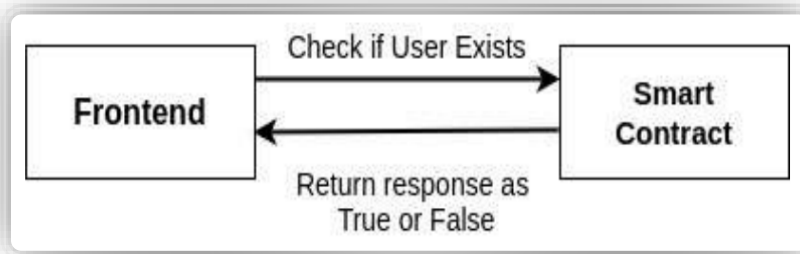
SYSTEM ARCHITECTURE



LIST OF MODULES

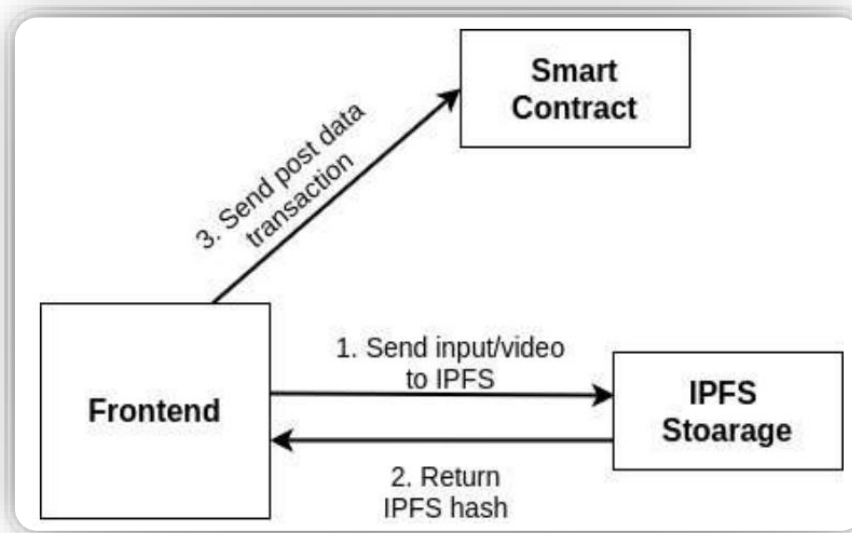
- ✓ Process for registering
- ✓ Process for creating post
- ✓ Reward System

EXPLANATION OF MODULES: Process of registering



When the client starts, the user can register into the application only if he/she is not already registered. This is handled by the users mapping and frontend. If the account address of user already exists in the users mapping, the frontend registration is disabled, else the frontend sends a transaction along with 0.002 Ether (the cryptocurrency of Ethereum blockchain) to the blockchain and a new user account is registered with the application. This amount goes into the contract balance and it enables the contract to reward the users based on the likes (or upvotes) their content gets. It also discourages malicious actors to get registered on the application.

EXPLANATION OF MODULES: Process of creating post



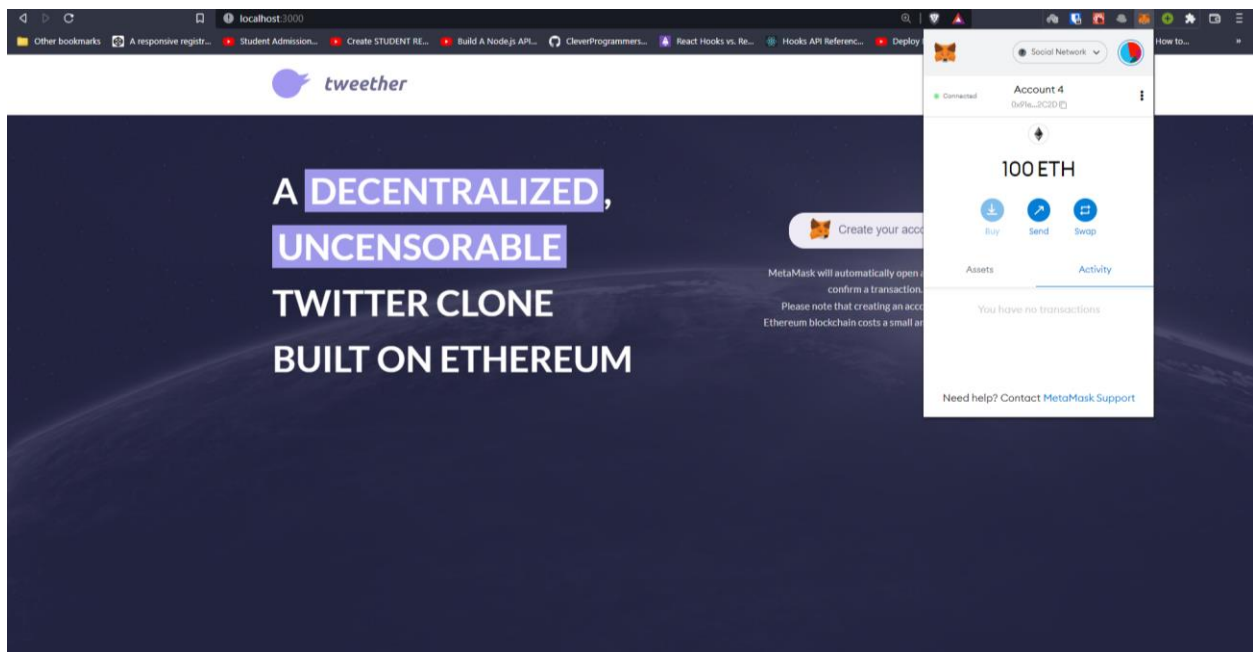
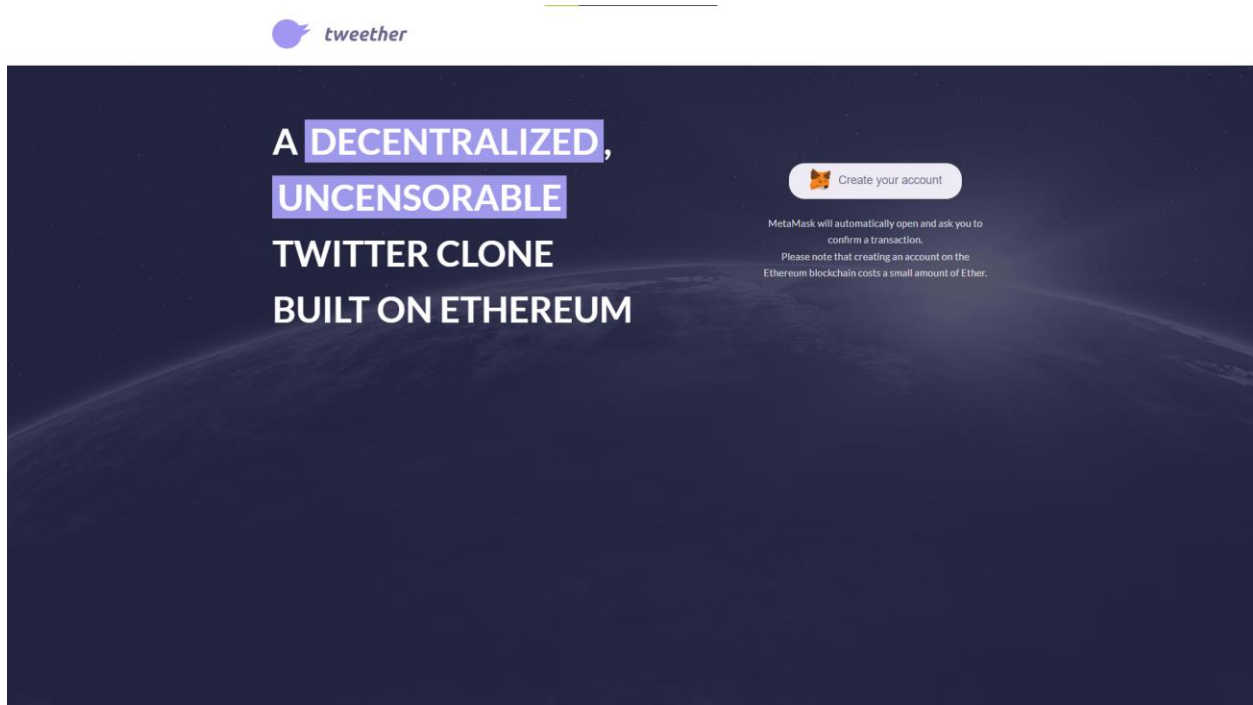
While creating a new post, if an image or video is uploaded by the user along with the post, the frontend client loads the image or video into a byte array and sends the byte data to IPFS storage endpoint through IPFS library. When the data storage is successful, it returns the corresponding IPFS hash in return. This hash is stored on the blockchain and is later used to retrieve the image or video when required. All the post data along with creator's address, image/video IPFS hash and current timestamp is sent as a transaction to the blockchain along with an amount of 0.001 Ether to the blockchain and the post is created. Other users can now like (upvote) or comment on the post.

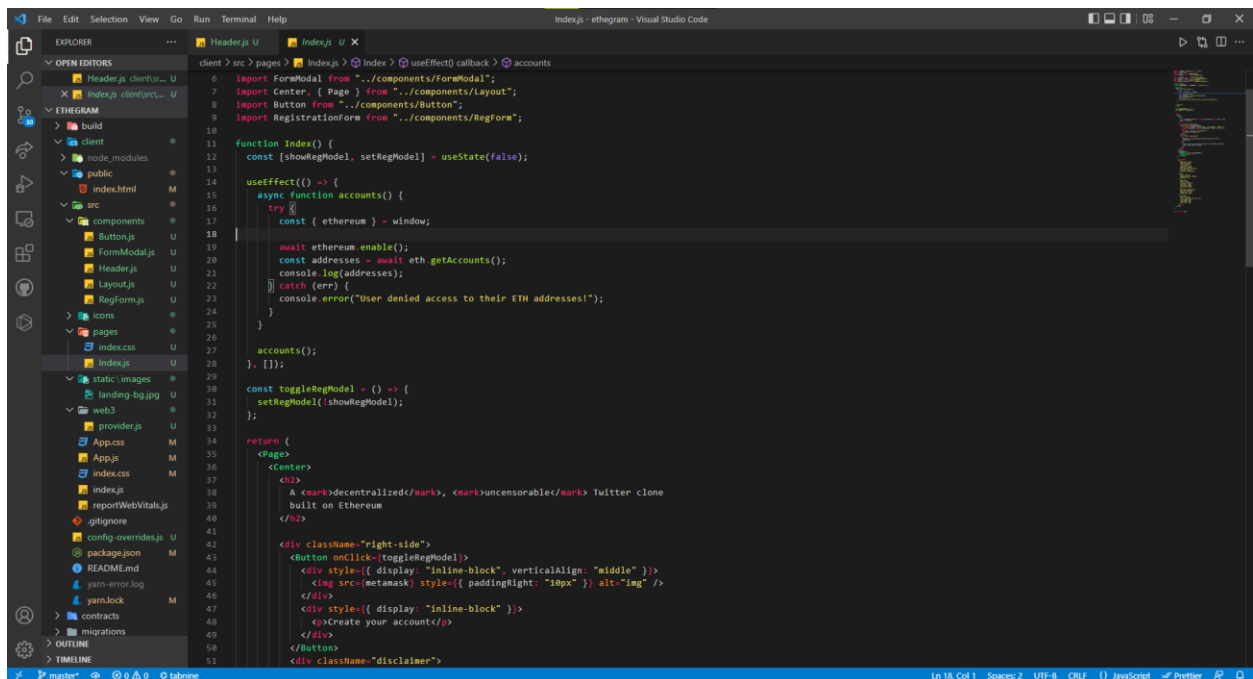
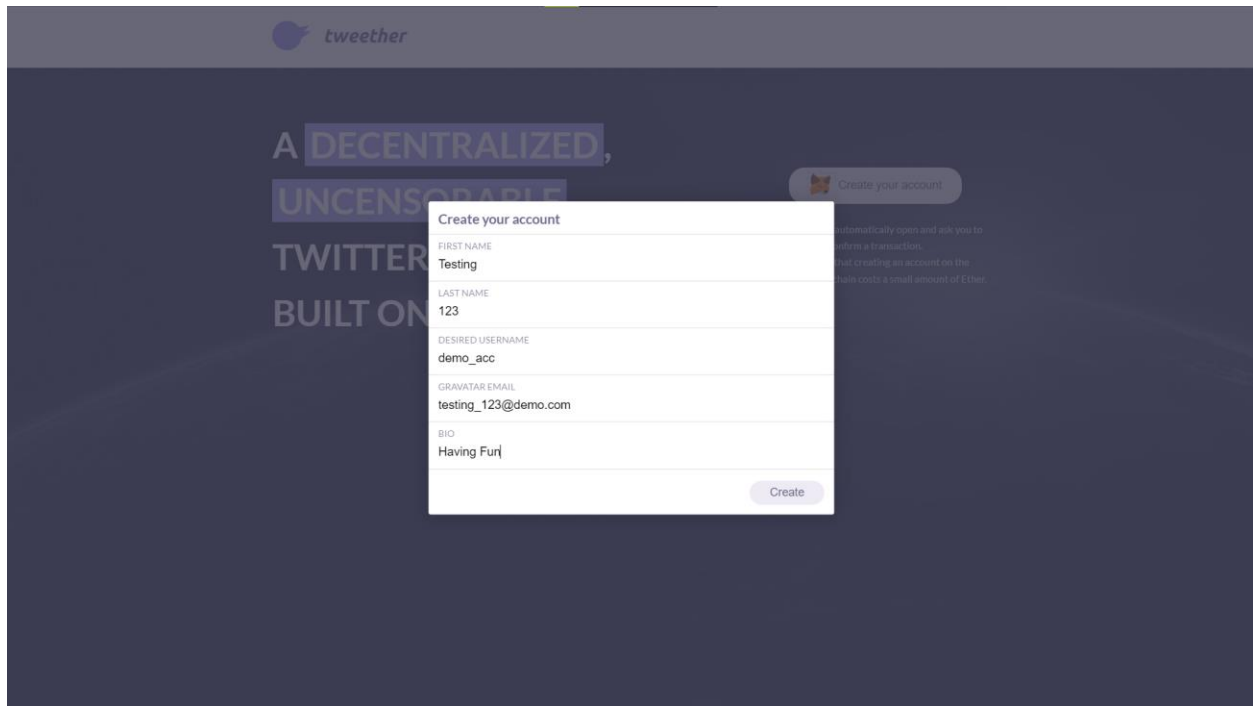
EXPLANATION OF MODULES: Reward system

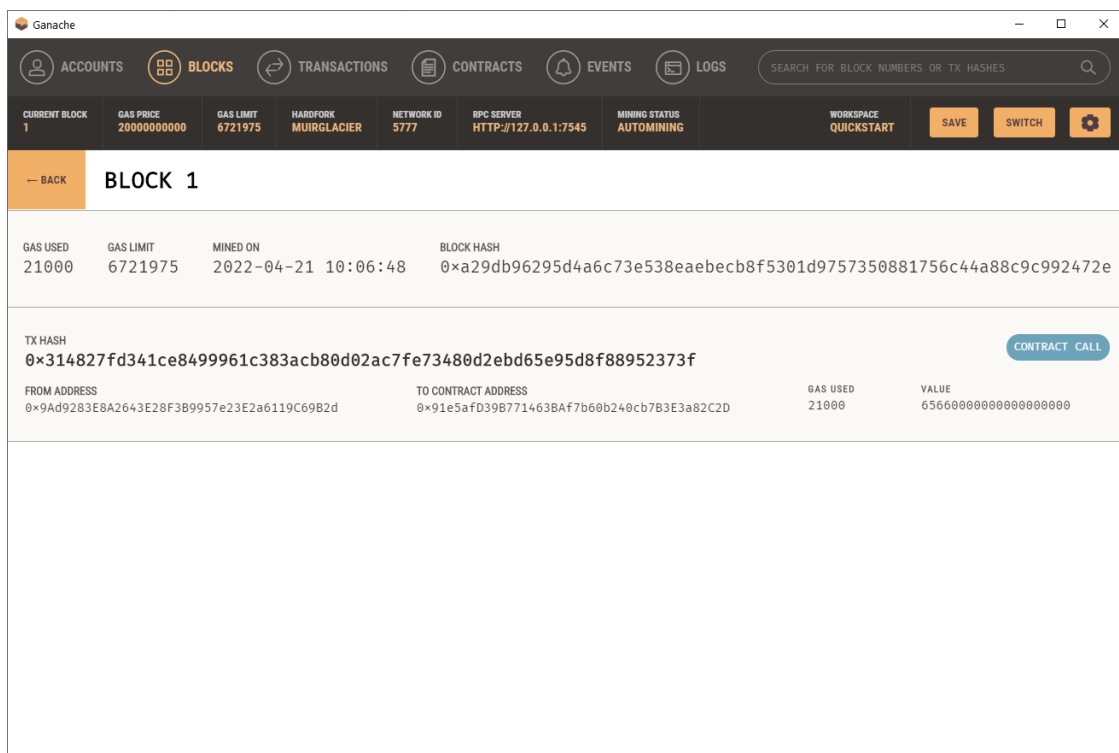
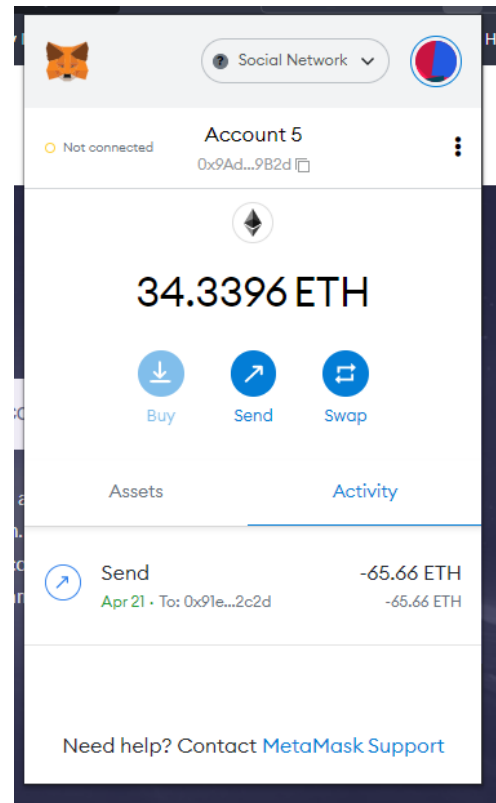
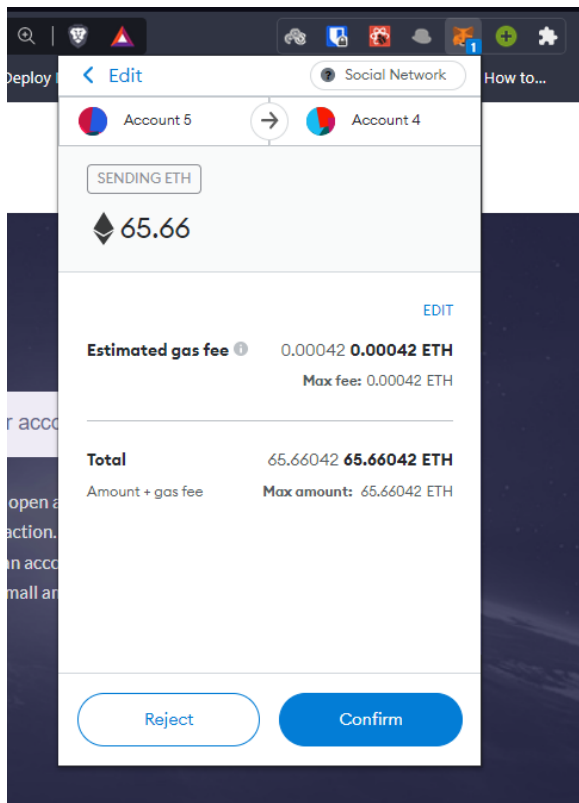
When the user creates a post, other users of the application can view and upvote the post if they like the content. Each upvote generates a single token in the application account of the creator. These tokens can be redeemed into Ethers from the platform itself. The funds are directly transferred to the account address from the contract balance. Each token when redeemed gives 0.0001 Ether. Thus, the cost of creating a new post is recovered when the user receives 10 upvotes on his post. Content creators are rewarded only based on how much their content is liked by other users of the application.

IMPEMENTATION

ETHEGRAM UI







STORAGE CONTRACTS:

The screenshot shows the Visual Studio Code interface with two Solidity contracts open. The left contract, `TweetStorage.sol`, defines a `Tweet` struct with fields for `uint256 id`, `string text`, `uint256 userId`, and `uint256 postedAt`. It includes a `mapping(uint256 => Tweet) public tweets` and a `uint256 latestTweetId`. A `createTweet` function is implemented, which increments `latestTweetId` and stores a new `Tweet` object. The right contract, `UserStorage.sol`, defines a `Profile` struct with `uint256 id` and `bytes32 username`. It includes a `mapping(uint256 => Profile) public profiles` and a `uint256 latestUserId`. A `createUser` function is implemented, which increments `latestUserId` and stores a new `Profile` object. The Explorer sidebar on the left shows the project structure, including `contracts`, `users`, `migrations`, `test`, and `integration` folders.

```
contracts > tweets > TweetStorage.sol
1  pragma solidity ^0.8.13;
2
3  contract TweetStorage {
4      mapping(uint256 => Tweet) public tweets;
5
6      struct Tweet {
7          uint256 id;
8          string text;
9          uint256 userId;
10         uint256 postedAt;
11     }
12
13     uint256 latestTweetId = 0;
14
15     function createTweet(uint256 _userId, string memory _text)
16     public
17     returns (uint256)
18     {
19         latestTweetId++;
20         tweets[latestTweetId] = Tweet(latestTweetId, _text, _userId, block.timestamp);
21         return latestTweetId;
22     }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
contracts > users > UserStorage.sol
1  pragma solidity ^0.8.13;
2
3  contract UserStorage {
4      mapping(uint256 => Profile) public profiles;
5
6      struct Profile {
7          uint256 id;
8          bytes32 username;
9      }
10
11     uint256 latestUserId = 0;
12
13     function createUser(bytes32 _username) public returns (uint256) {
14         latestUserId++;
15         profiles[latestUserId] = Profile(latestUserId, _username);
16         return latestUserId;
17     }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

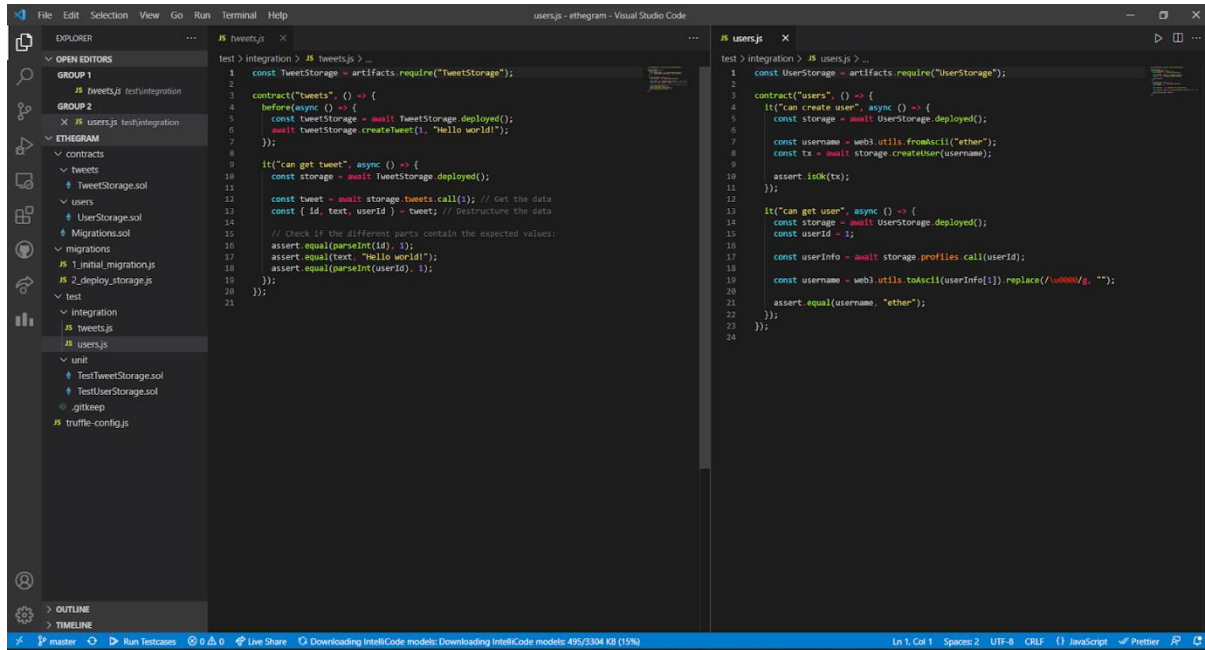
UNIT TESTING:

The screenshot shows the Visual Studio Code interface with two Solidity test files open. The left test file, `TestTweetStorage.sol`, contains a `testCreateTweet` function that uses `TruffleAssert` to verify that a tweet is created with the correct ID and text. The right test file, `TestUserStorage.sol`, contains a `testCreateUser` function that uses `TruffleAssert` to verify that a user is created with the correct ID and username. The Explorer sidebar on the left shows the project structure, including `contracts`, `users`, `migrations`, `test`, and `integration` folders.

```
test > unit > TestTweetStorage.sol
1  pragma solidity ^0.8.13;
2
3  import "truffle/Assert.sol";
4  import "truffle/DeployedAddresses.sol";
5  import "../contracts/tweets/TweetStorage.sol";
6
7  contract TestTweetStorage {
8      function testCreateTweet() public {
9          TweetStorage _storage = TweetStorage(DeployedAddresses.TweetStorage());
10
11         uint256 _userId = 1;
12         uint256 _expectedTweetId = 1;
13
14         Assert.equal(
15             _storage.createTweet(_userId, "Hello world"),
16             _expectedTweetId,
17             "Should create tweet with ID 1"
18         );
19     }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
test > unit > TestUserStorage.sol
1  pragma solidity ^0.8.13;
2
3  import "truffle/Assert.sol";
4  import "truffle/DeployedAddresses.sol";
5  import "../contracts/users/UserStorage.sol";
6
7  contract TestUserStorage {
8      function testCreateUser() public {
9          // use the deployed contract
10         UserStorage _storage = UserStorage(DeployedAddresses.UserStorage());
11
12         uint256 _expectedId = 1;
13
14         Assert.equal(
15             _storage.createUser("ether"),
16             _expectedId,
17             "Should create user with ID 1"
18         );
19     }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

INTEGRATION TESTING:

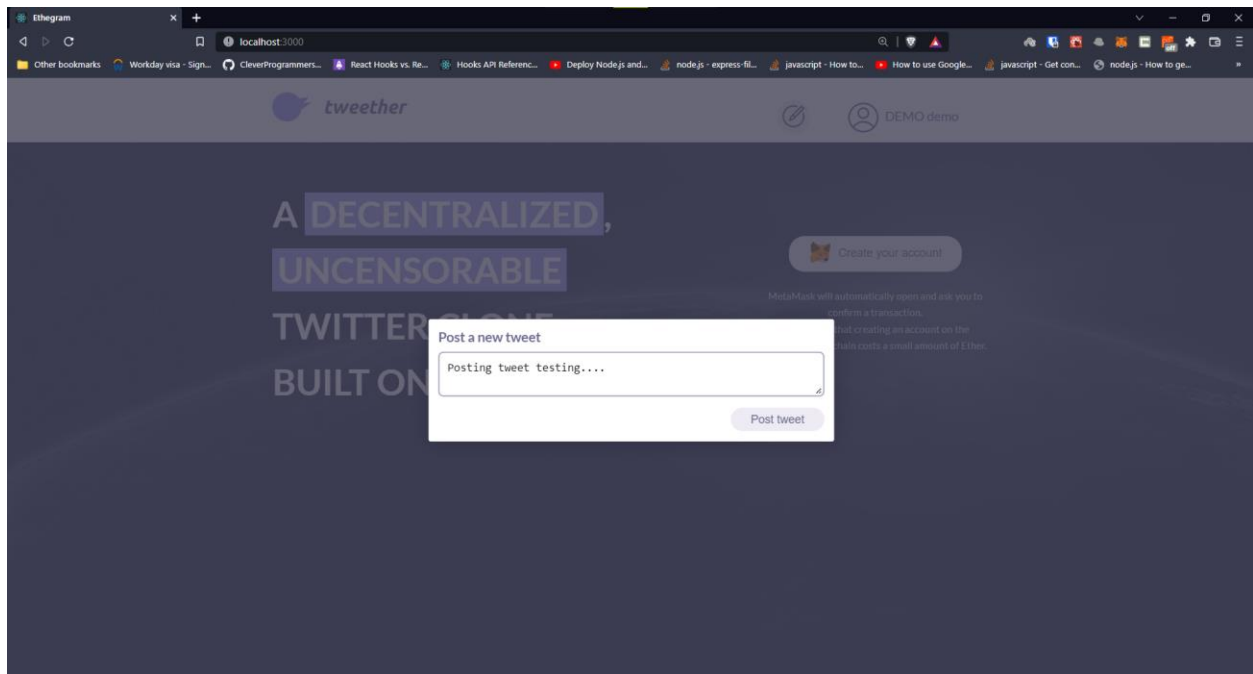


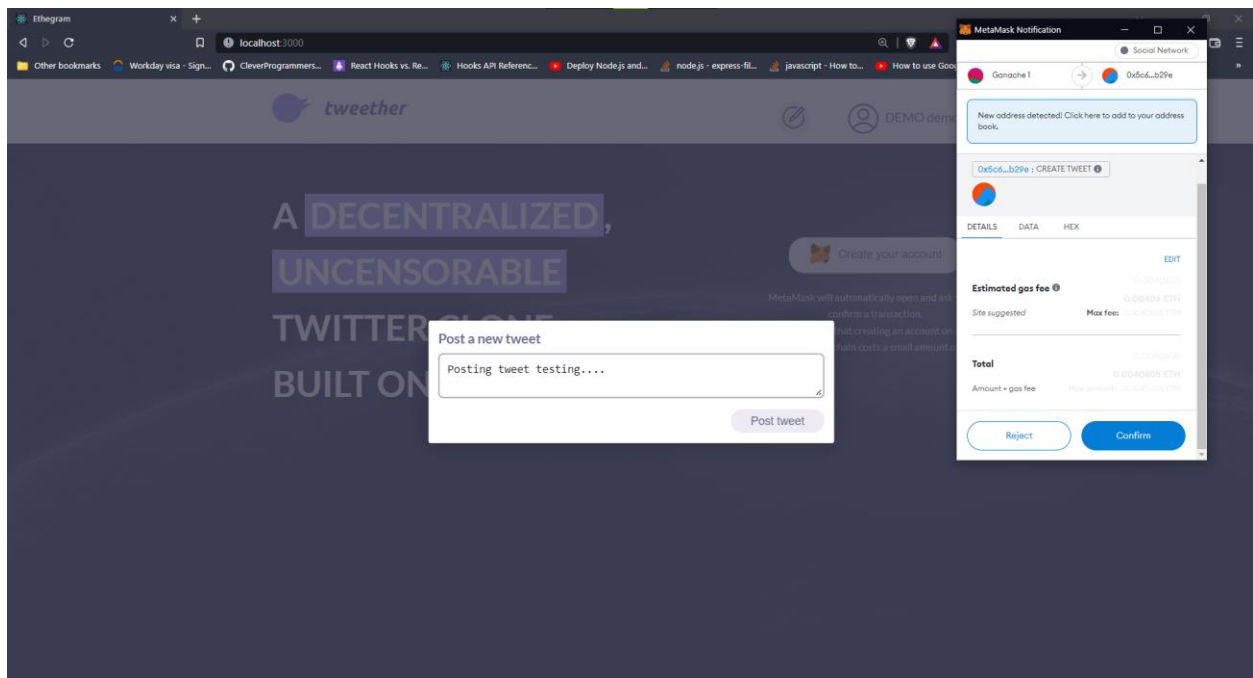
The screenshot shows the Visual Studio Code interface with two files open: `tweets.js` and `users.js`. The Explorer on the left shows the project structure for 'ethgram', including contracts, migrations, and test files. The status bar at the bottom indicates the current file is `Ln 1, Col 1` with a UTF-8 encoding and LF line endings.

```
test > integration > JS tweets.js > ...
1  const TweetStorage = artifacts.require("TweetStorage");
2
3  contract("tweets", () => {
4    beforeEach(async () => {
5      const tweetStorage = await TweetStorage.deployed();
6      await tweetStorage.createTweet(1, "Hello world!");
7    });
8
9    it("can get tweet", async () => {
10     const storage = await tweetStorage.deployed();
11
12     const tweet = await storage.tweets.call(1); // Get the data
13     const { id, text, userId } = tweet; // Destructure the data
14
15     // Check if the different parts contain the expected values:
16     assert.equal(parseInt(id), 1);
17     assert.equal(text, "Hello world!");
18     assert.equal(parseInt(userId), 1);
19   });
20 });
21
```

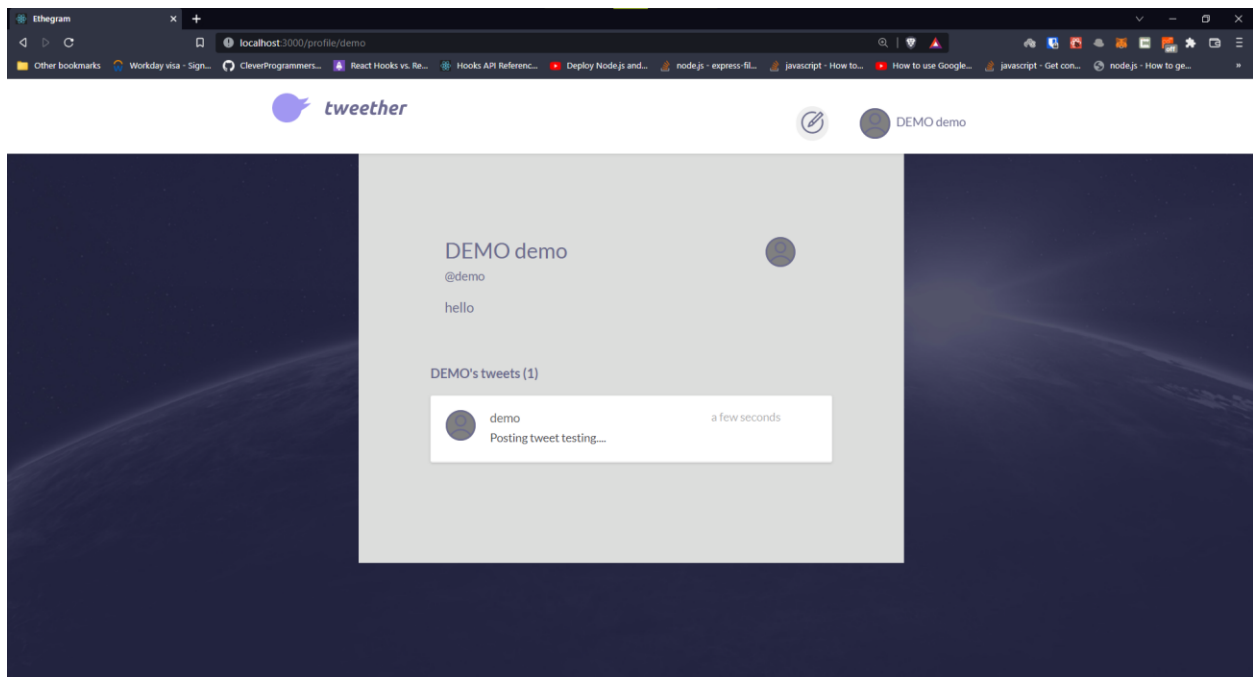
```
test > integration > JS users.js > ...
1  const UserStorage = artifacts.require("UserStorage");
2
3  contract("users", () => {
4    it("can create user", async () => {
5      const storage = await UserStorage.deployed();
6
7      const username = web3.utils.fromAscii("ether");
8      const tx = await storage.createUser(username);
9
10     assert.isOk(tx);
11   });
12
13   it("can get user", async () => {
14     const storage = await UserStorage.deployed();
15     const userId = 1;
16
17     const userInfo = await storage.profiles.call(userId);
18     const username = web3.utils.toAscii(userInfo[1].replace(/^u0000/g, ""));
19
20     assert.equal(username, "ether");
21   });
22 });
23
24
```

POSTING TWEETS

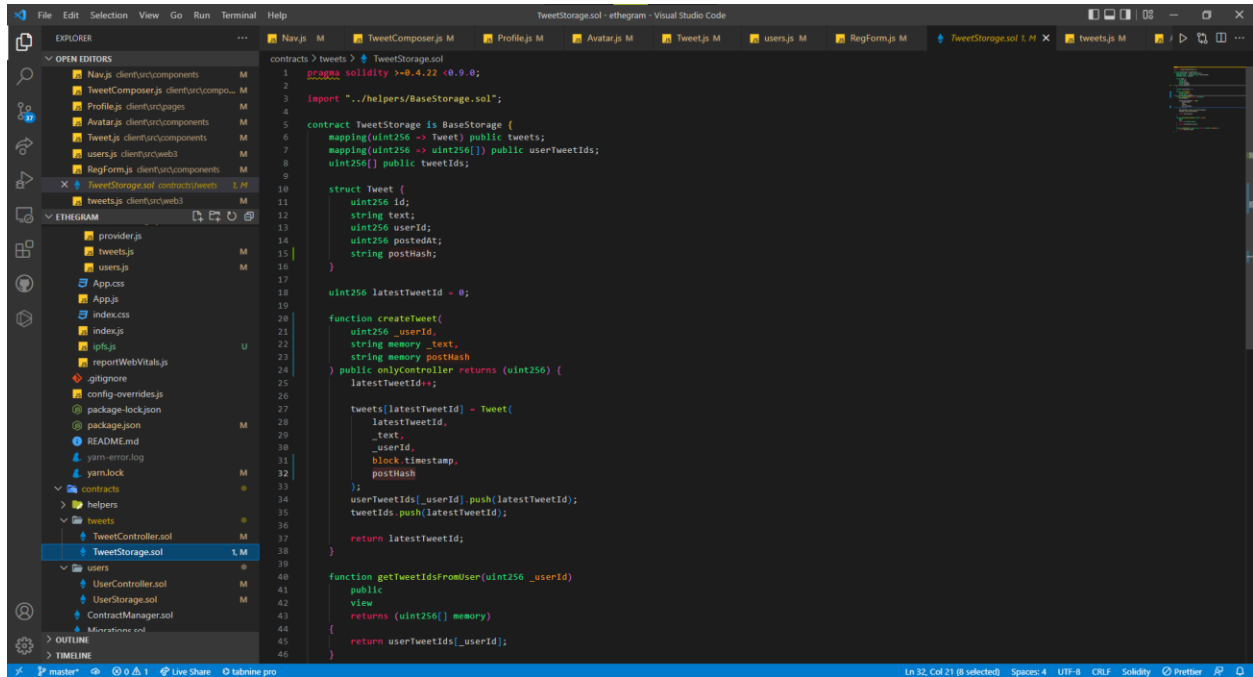




PROFILE PAGE



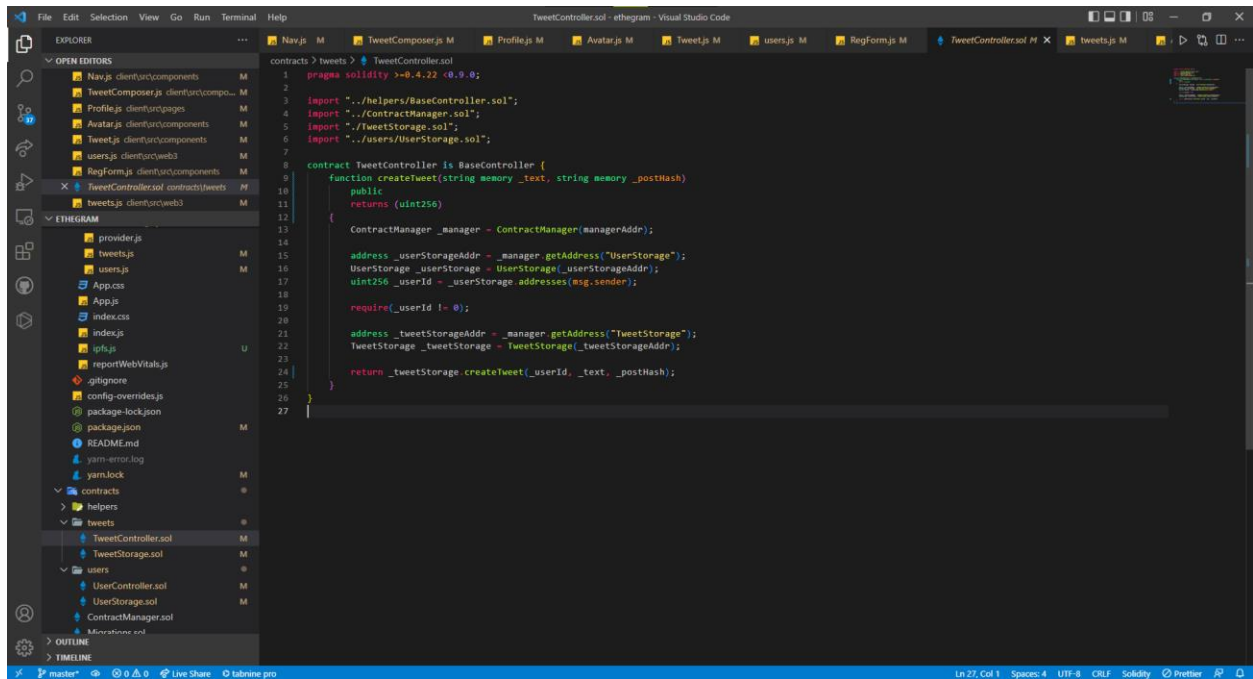
TWEET STORAGE CONTRACT (updt.)



The screenshot shows the TweetStorage.sol contract in a Solidity environment. The code defines a contract that inherits from BaseStorage and implements a storage system for tweets. It includes a mapping for tweets, a mapping for user tweet IDs, and a struct for the Tweet object. The createTweet function is implemented to store a new tweet and return its ID.

```
contracts > tweets > TweetStorage.sol
1  pragma solidity >=0.4.22 <0.9.0;
2
3  import "../helpers/BaseStorage.sol";
4
5  contract TweetStorage is BaseStorage {
6      mapping(uint256 => Tweet) public tweets;
7      mapping(uint256 => uint256[]) public userTweetIds;
8      uint256[] public tweetIds;
9
10     struct Tweet {
11         uint256 id;
12         string text;
13         uint256 userId;
14         uint256 postDate;
15         string postHash;
16     }
17
18     uint256 latestTweetId = 0;
19
20     function createTweet(
21         uint256 _userId,
22         string memory _text,
23         string memory _postHash
24     ) public onlyController returns (uint256) {
25         latestTweetId++;
26
27         tweets[latestTweetId] = Tweet(
28             latestTweetId,
29             _text,
30             _userId,
31             block.timestamp,
32             _postHash
33         );
34         userTweetIds[_userId].push(latestTweetId);
35         tweetIds.push(latestTweetId);
36
37         return latestTweetId;
38     }
39
40     function getTweetIdsFromUser(uint256 _userId)
41     public
42     view
43     returns (uint256[] memory)
44     {
45         return userTweetIds[_userId];
46     }
47 }
```

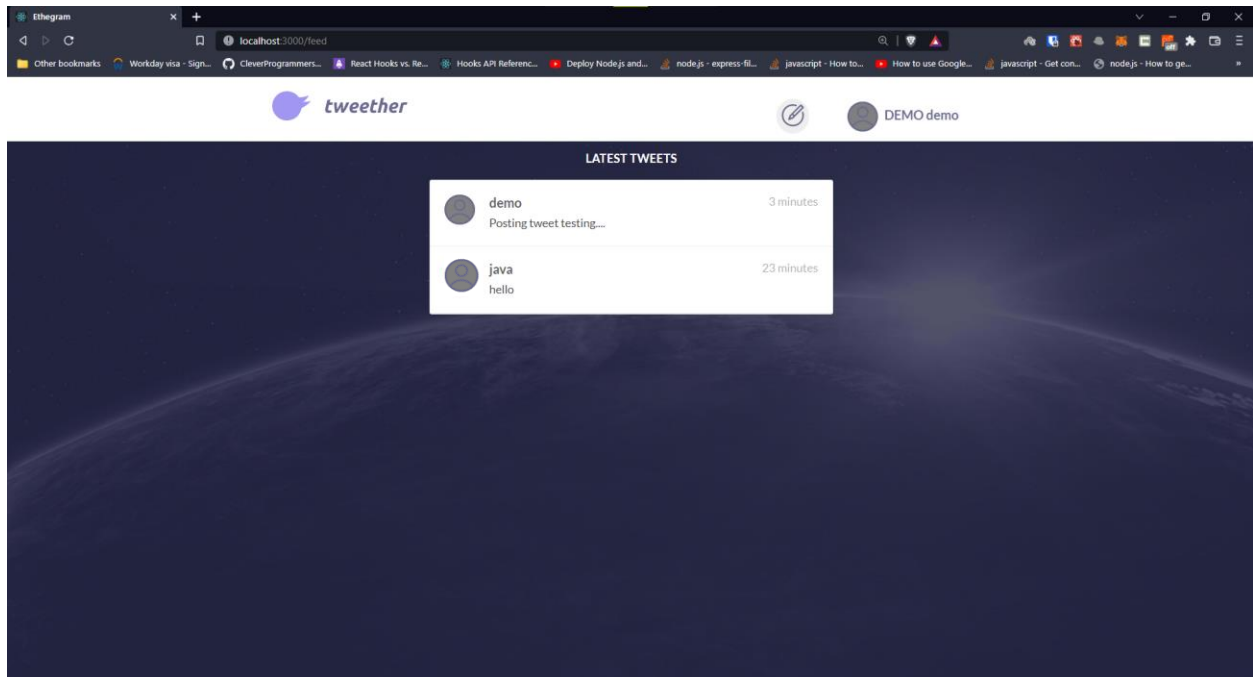
TWEET CONTROLLER



The screenshot shows the TweetController.sol contract in a Solidity environment. The code defines a contract that inherits from BaseController and implements a controller for the TweetStorage contract. It includes a createTweet function that interacts with the TweetStorage contract to create a new tweet.

```
contracts > tweets > TweetController.sol
1  pragma solidity >=0.4.22 <0.9.0;
2
3  import "../helpers/BaseController.sol";
4  import "../ContractManager.sol";
5  import "../TweetStorage.sol";
6  import "../users/UserStorage.sol";
7
8  contract TweetController is BaseController {
9      function createTweet(string memory _text, string memory _postHash)
10     public
11     returns (uint256)
12     {
13         ContractManager _manager = ContractManager(managerAddr);
14
15         address _userStorageAddr = _manager.getAddress("UserStorage");
16         UserStorage _userStorage = UserStorage(_userStorageAddr);
17         uint256 _userId = _userStorage.addresses(msg.sender);
18
19         require(_userId != 0);
20
21         address _tweetStorageAddr = _manager.getAddress("TweetStorage");
22         TweetStorage _tweetStorage = TweetStorage(_tweetStorageAddr);
23
24         return _tweetStorage.createTweet(_userId, _text, _postHash);
25     }
26
27 }
```

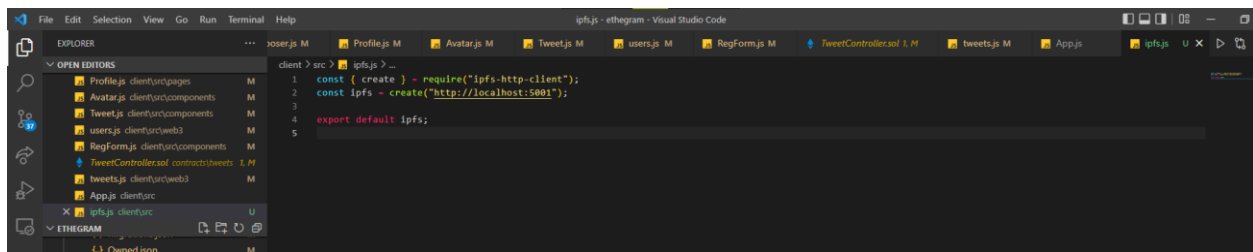
FEED PAGE



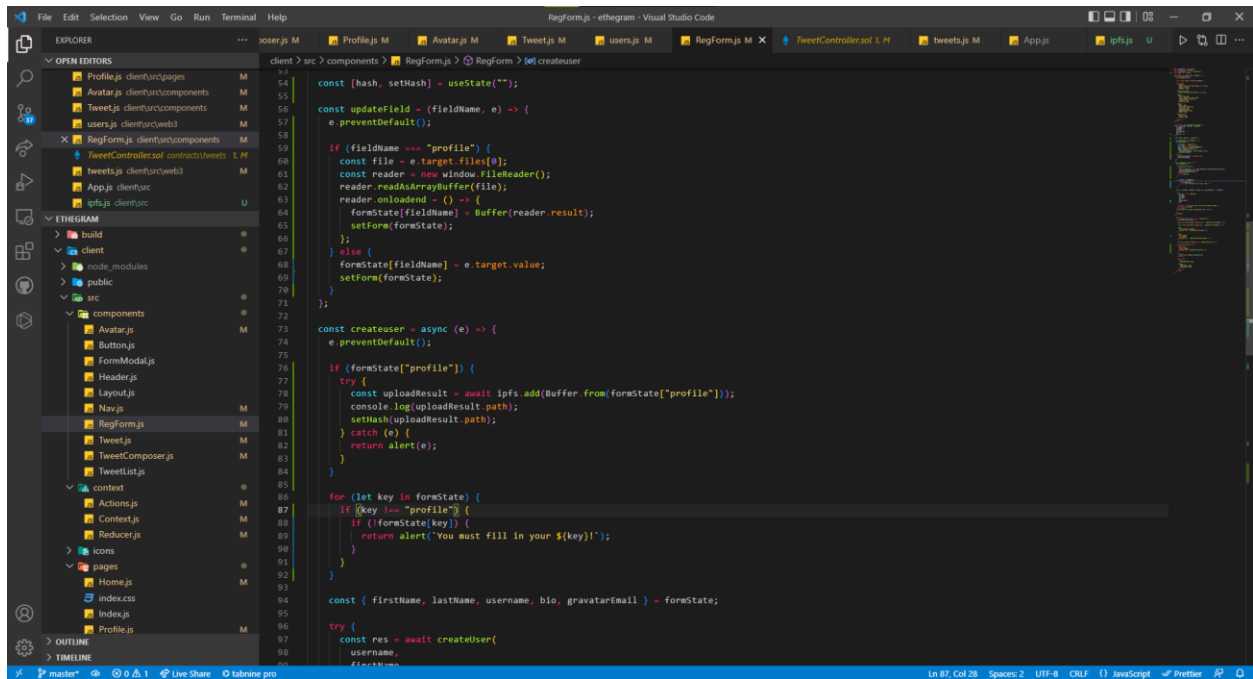
CONNECTING APPLICATION WITH IPFS

```
Command Prompt - ipfs: daemon

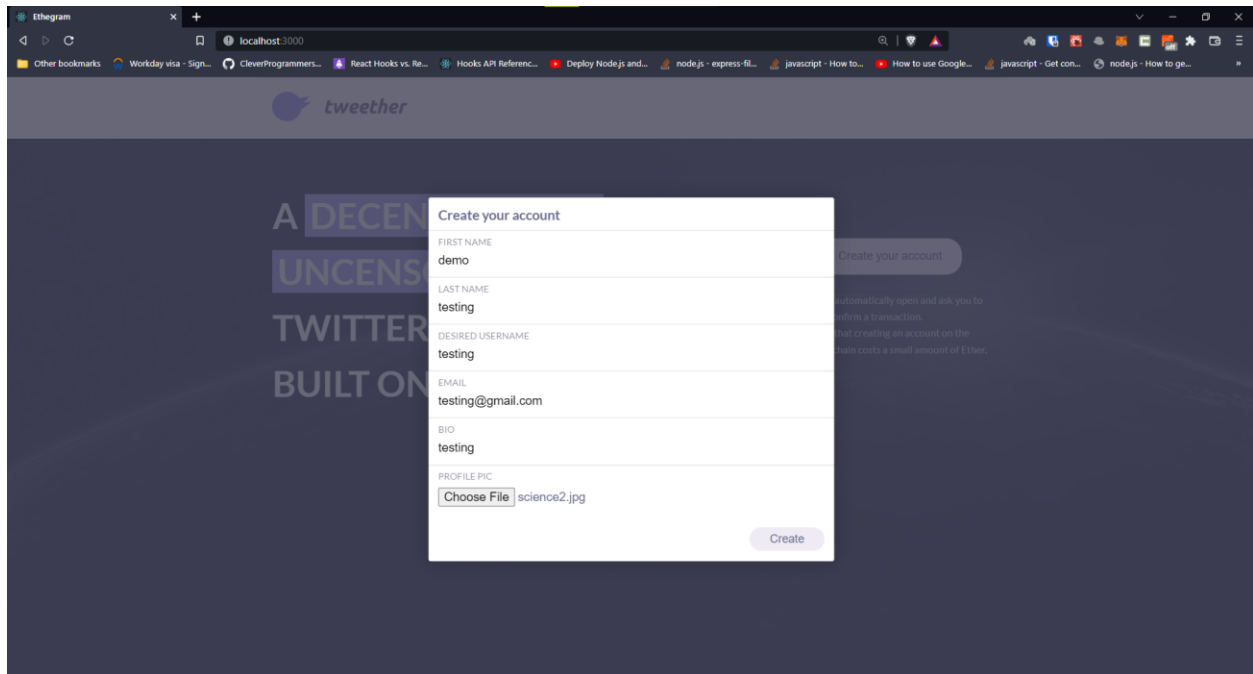
C:\Users\versan>ipfs daemon
Initializing daemon...
go-ipfs version: 0.12.2
Repo version: 12
System version: amd64/windows
Golang version: go1.16.15
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/127.0.0.1/udp/4001/quic
Swarm listening on /ip4/169.254.159.97/tcp/4001
Swarm listening on /ip4/169.254.237.169/tcp/4001
Swarm listening on /ip4/169.254.237.169/udp/4001/quic
Swarm listening on /ip4/169.254.95.198/tcp/4001
Swarm listening on /ip4/169.254.95.198/udp/4001/quic
Swarm listening on /ip6:::1/tcp/4001
Swarm listening on /ip6:::1/udp/4001/quic
Swarm listening on /p2p-circuit
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/127.0.0.1/udp/4001/quic
Swarm announcing /ip4/169.254.159.97/tcp/4001
Swarm announcing /ip4/169.254.159.97/udp/4001/quic
Swarm announcing /ip4/169.254.237.169/tcp/4001
Swarm announcing /ip4/169.254.237.169/udp/4001/quic
Swarm announcing /ip4/169.254.95.198/tcp/4001
Swarm announcing /ip4/169.254.95.198/udp/4001/quic
Swarm announcing /ip6:::1/tcp/4001
Swarm announcing /ip6:::1/udp/4001/quic
API server listening on /ip4/127.0.0.1/tcp/5001
WebUI: http://127.0.0.1:5001/webui
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
2022-05-19T09:50:32.797+0530  +[31mERROR-[0m  reprovider.simple    simple/reprovide.go:109 failed to reprovide: failed to find any peer in table
```

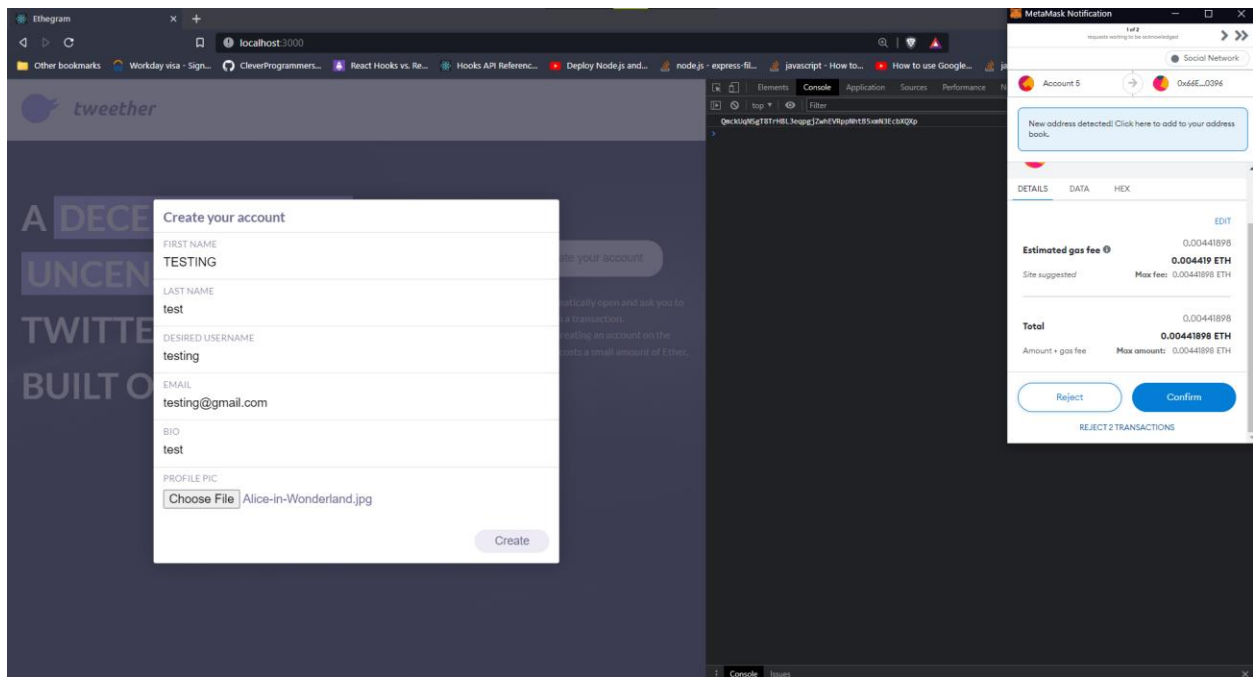
SENDING FILE BUFFER TO IPFS



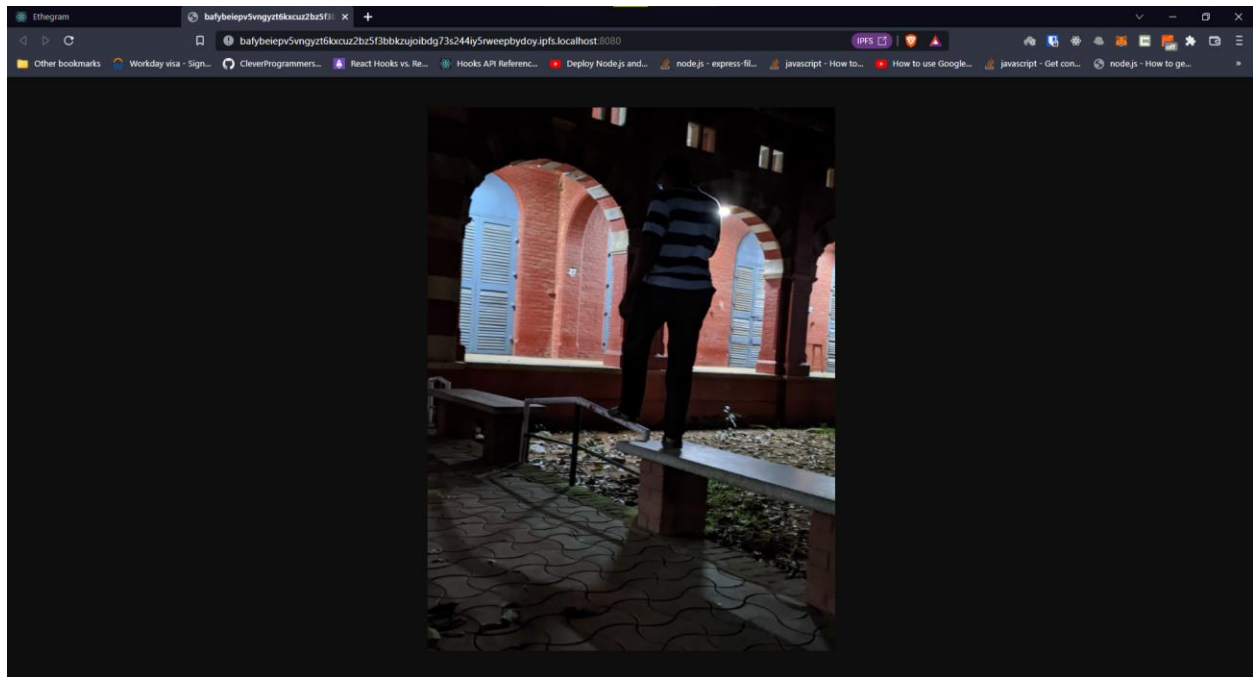
CHOOSING PROFILE PIC



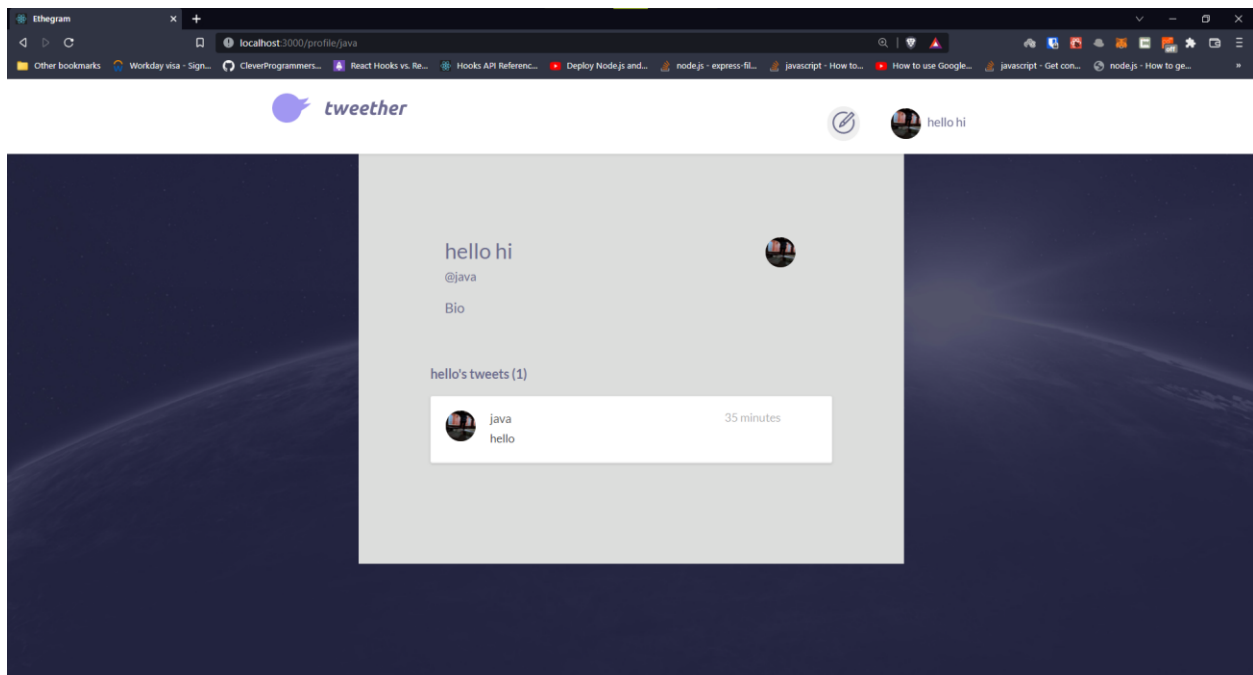
IPFS FILE HASH



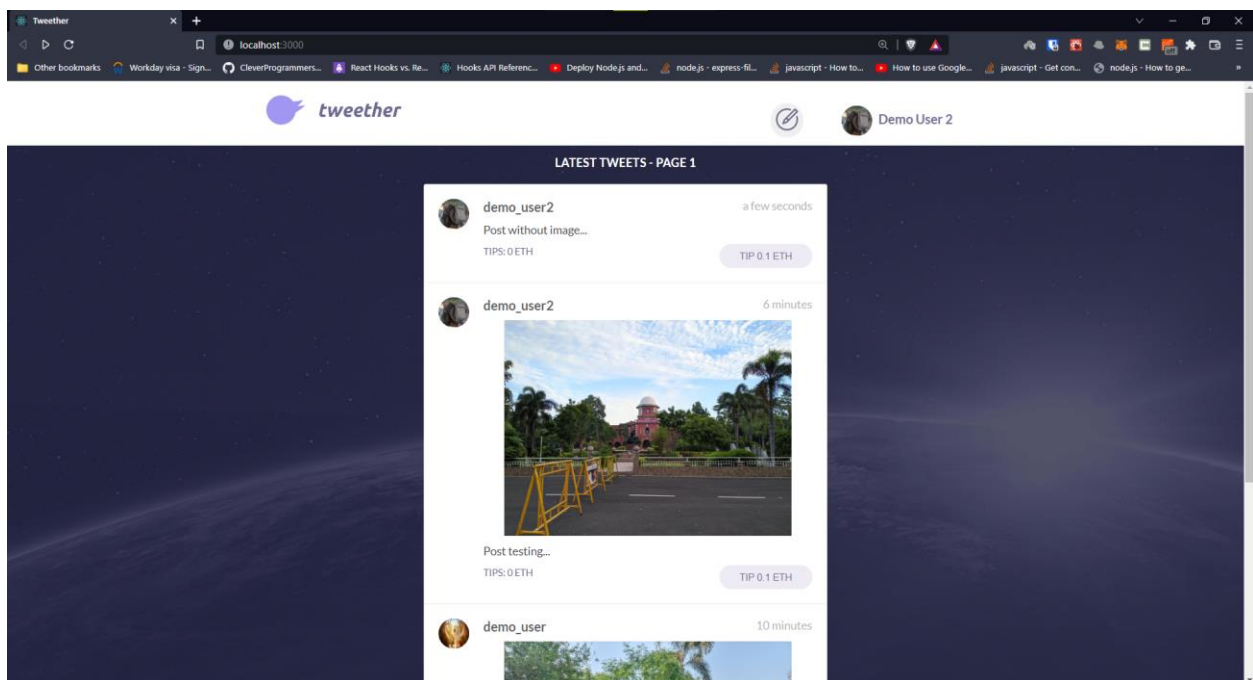
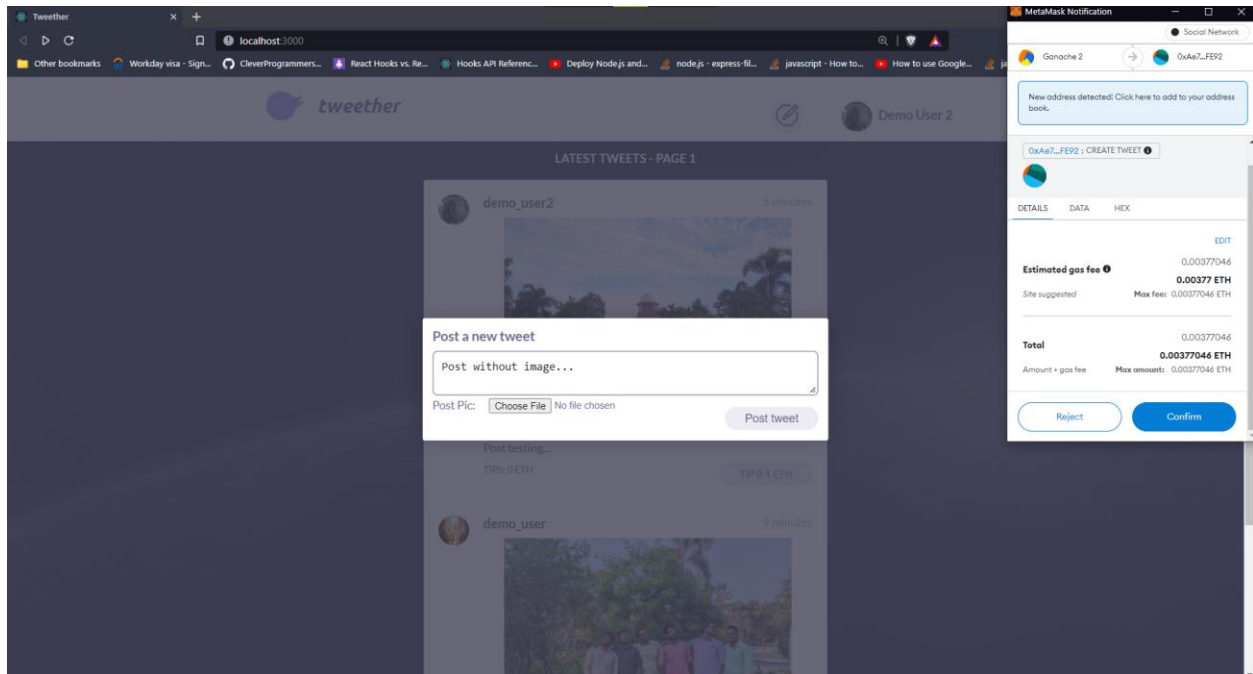
UPLOADED IMAGE SOURCE

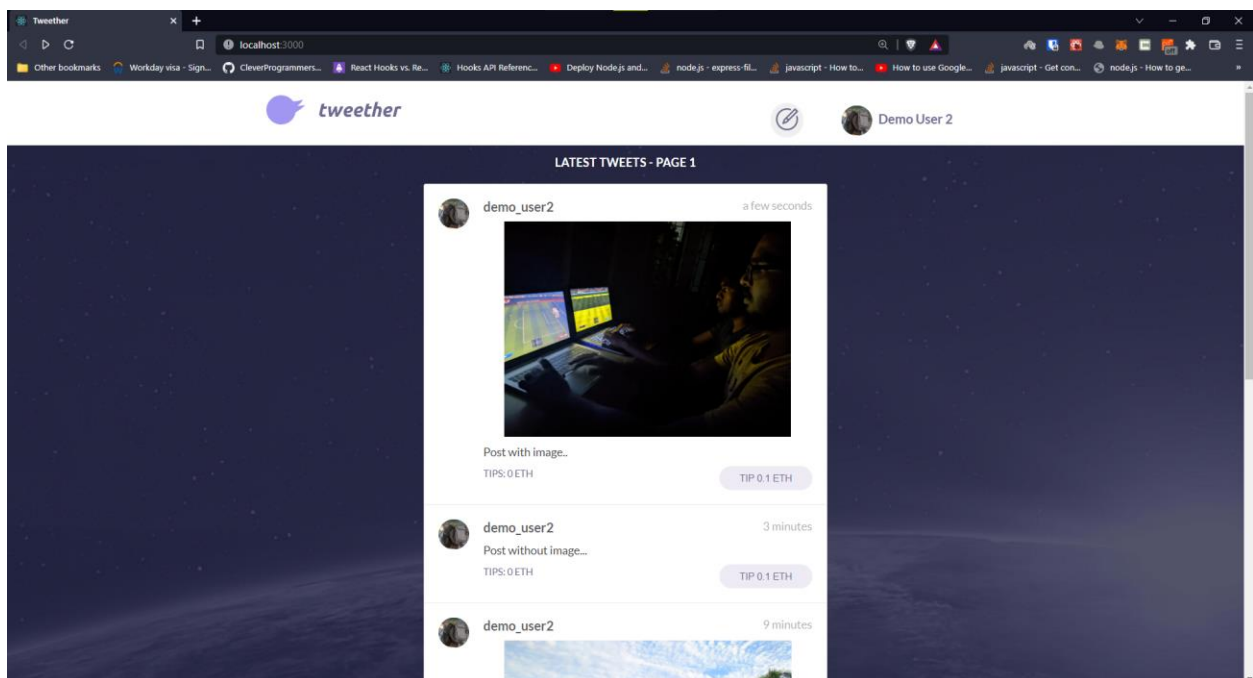
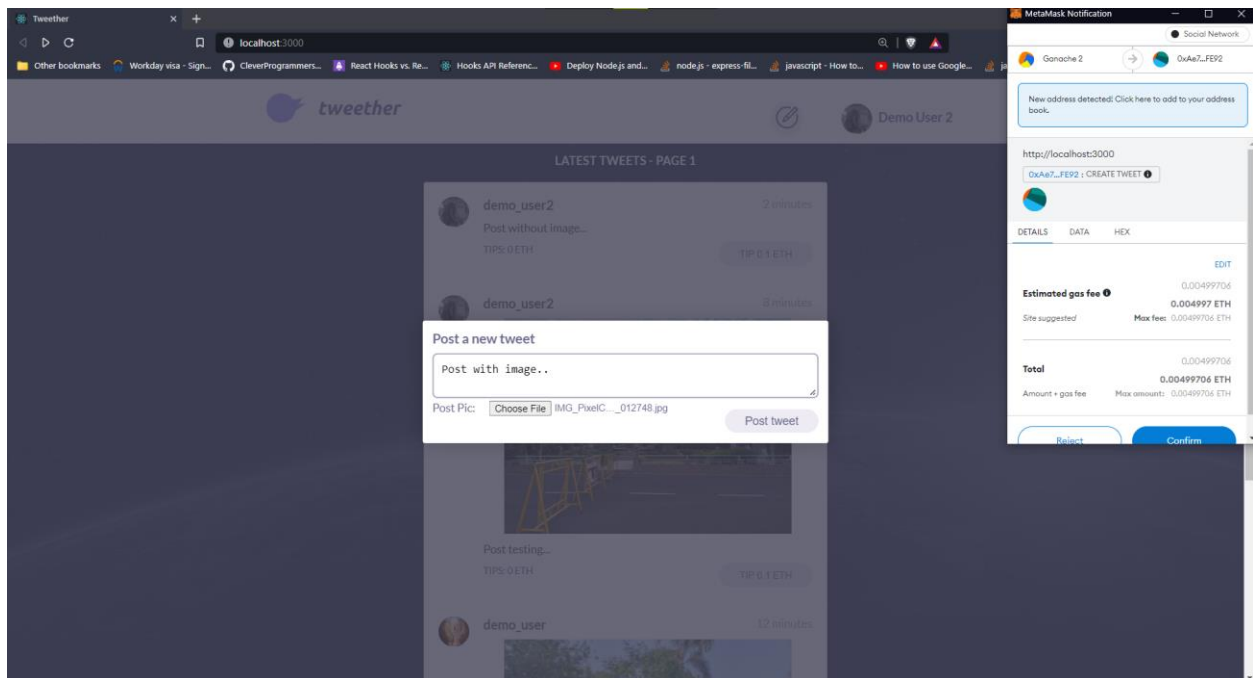


UPLOADED IMAGE AS PROFILE PIC



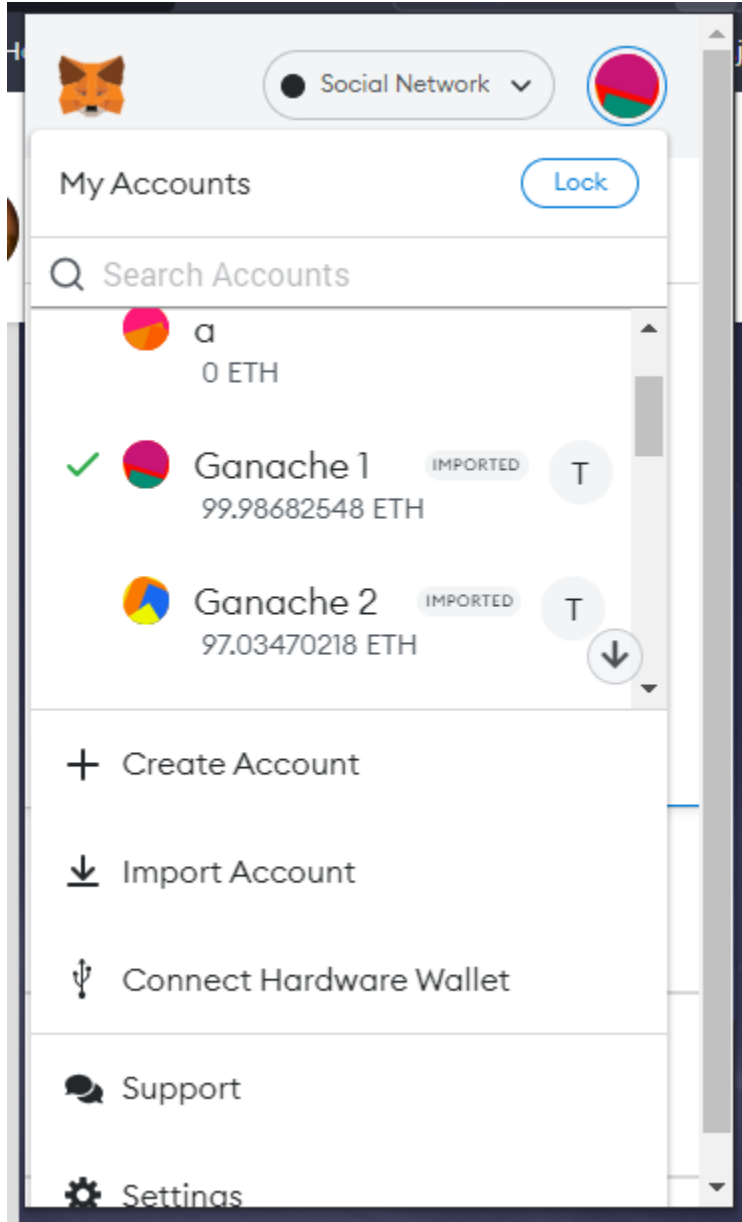
CREATE POST WITH AND WITHOUT IMAGE



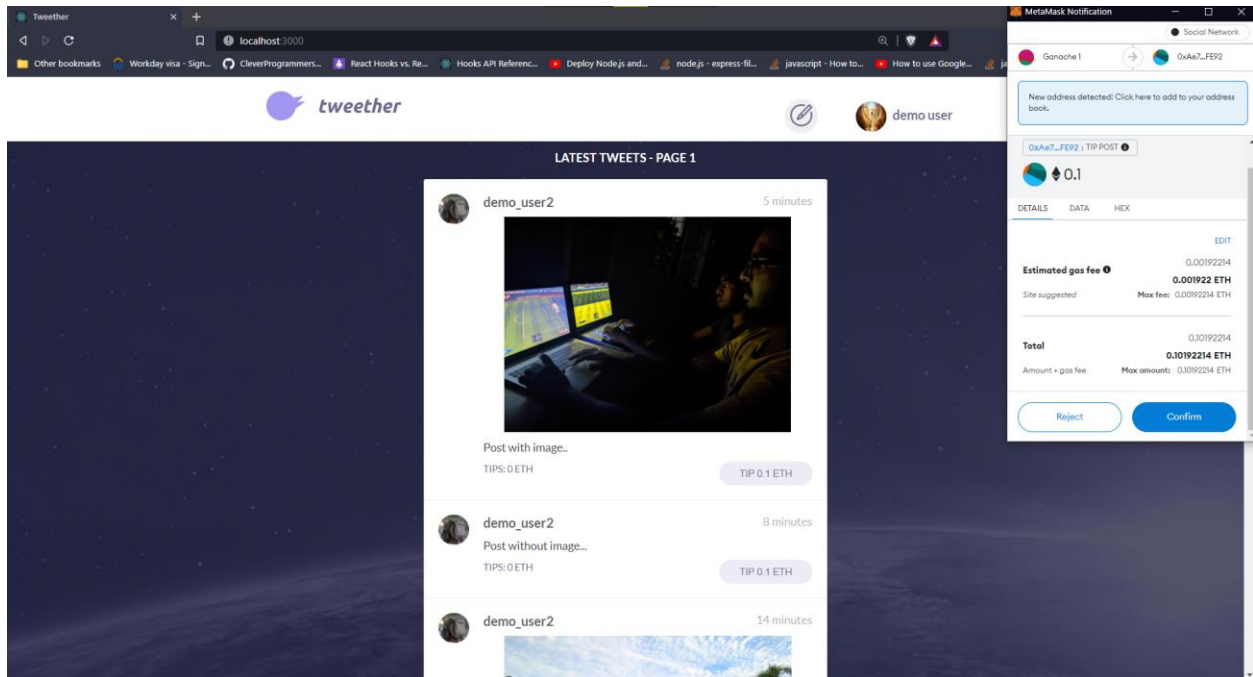


TIPPING FOR POSTS

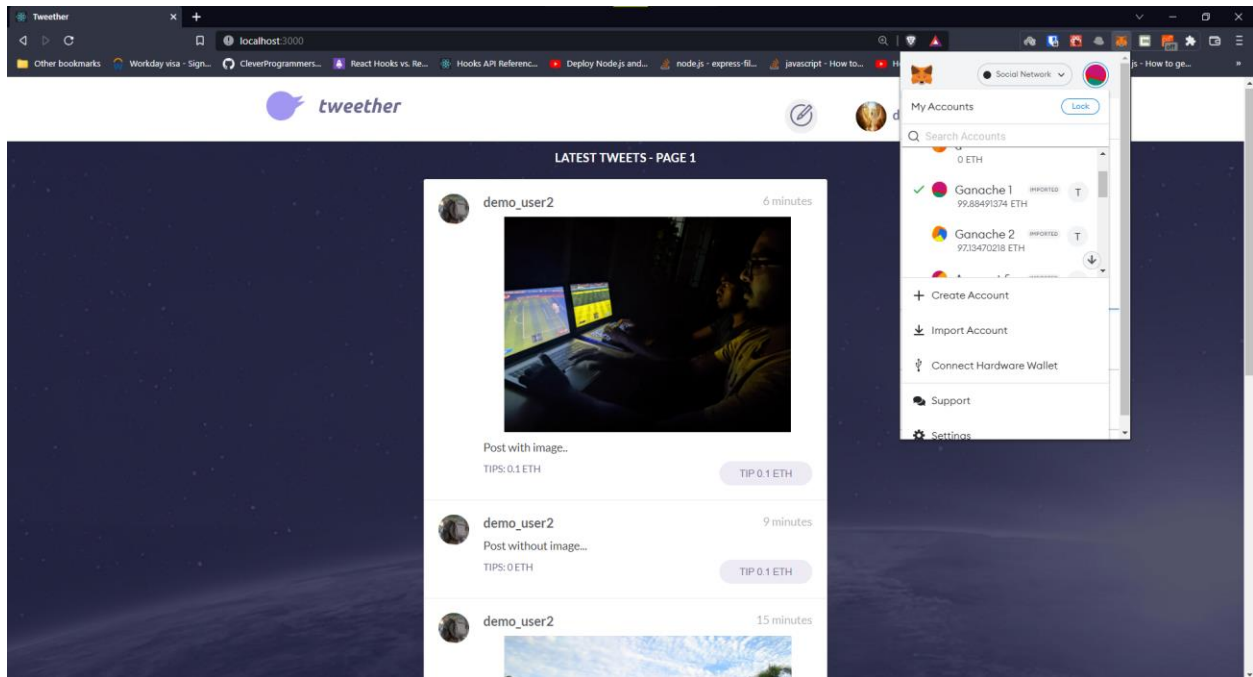
INITIAL BALANCE OF demo user (Ganache 1) and demo user2 (Ganache 2)



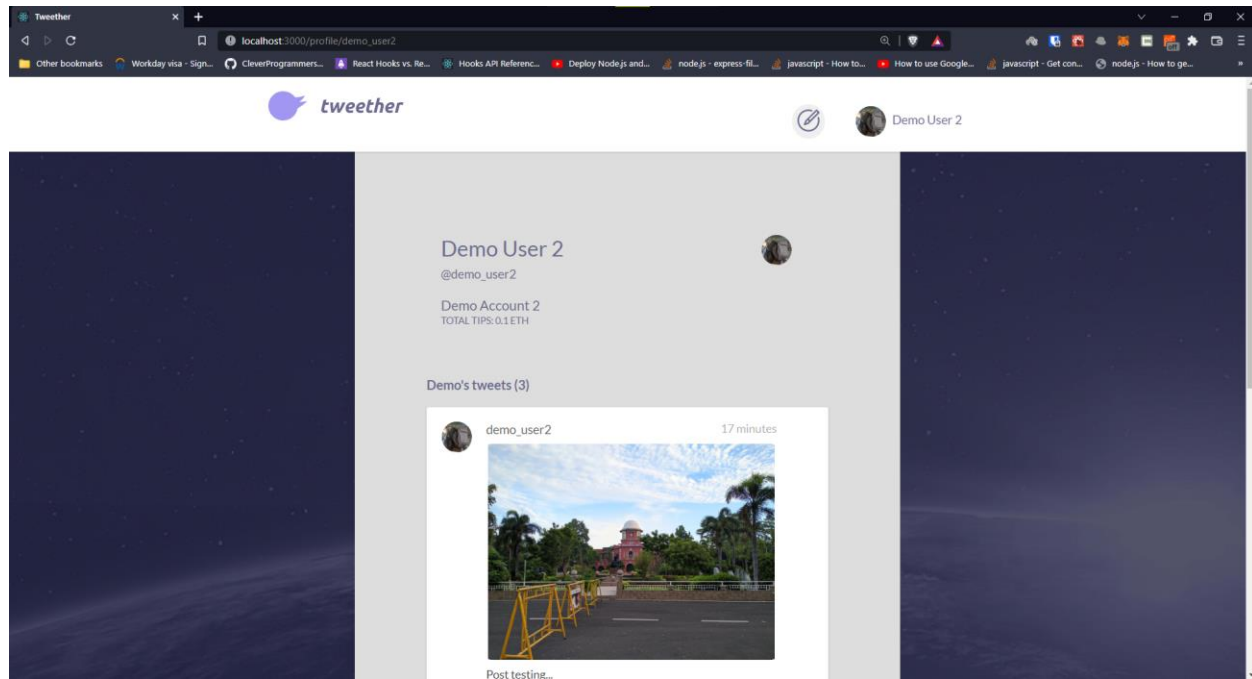
demo user TIPS 01.ETH TO POST CREATED BY demo user2



0.1 ETH GOT TRANSFERED FROM WALLET OF demo user TO demo user2



PROFILE PAGE



CONCLUSION:

As a result, a decentralised social network has been devised and implemented, which answers the questions highlighted previously. As a result, only the users control the content they upload in our Tweether application. Everyone's data is just 'out there,' in encrypted blobs that anybody may host or download but that only friends can decipher. Decentralization also makes it resistant to censorship, internet disruptions, and potential social monopolies. The decentralised social network paradigm has not only security, which is not difficult with public key cryptography, but also user-friendly security, which allows us to have the conveniences we've come to expect from centralised systems while keeping the network secure and open to anyone who wants to interact with it in any way they want.

REFERENCES:

- [1] (2018) Facebook Cambridge Analytica data scandal. [Online]. Available: https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal.
- [2] Quanqing Xu, Zhiwen Song, Rick Siow Mong Goh, Yongjun Li, "Building an Ethereum and IPFS-based Decentralized Social Network System", 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS).
- [3] Q. Xu, C. Jin, M. F. B. M. Rasid, B. Veeravalli, and K. M. M. Aung, "Blockchain-based decentralized content trust for docker images", *Multimedia Tools and Applications*, pp. 1–26, 2017.
- [4] (2018)Ethereum project. [Online]. Available: <https://www.ethereum.org/>.
- [5] (2018) Interplanetary file system. [Online]. Available: <https://ipfs.io/>.
- [6] Q. Xu, K. M. M. Aung, Y. Zhu, and K. L. Yong, "A blockchain-based storage system for data analytics in the internet of things," in *New Advances in the Internet of Things*. Springer, 2018, pp. 119–138.
- [7] A. Tar. (July 26 2018) Smart contracts, explained. [Online]. Available: <https://cointelegraph.com/explained/smart-contracts-explained>.
- [8] (2018) Akasha project. [Online]. Available: <https://akasha.world/>.
- [9] Antorweep Chakravorty and Chunming Rong, "Ushare: user controlled social media based on blockchain", *Conference Paper – January 2017*