

# **Blockchain: DApp Project Report**

## **on**

Yield Farming Staking App

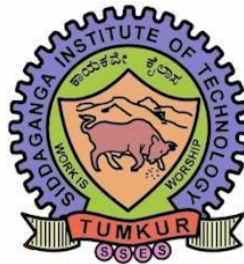
Submitted for the partial fulfillment of Bachelor of Engineering

By

- 1. Mohammed Sayeed (1SI18CS057)**
- 2. Chethan AR (1SI18CS027)**
- 3. Chetan CK(1SI18CS026)**

**Under the guidance of**

Dr. Nirmala M.B  
Associate Professor  
Dept. of CSE, SIT



**Department of Computer Science and Engineering**

**Siddaganga Institute of Technology, Tumakuru – 572103**

(An Autonomous Institution, affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)

**2021 -2022**

# Table Of Contents

- Introduction
- Problem Statement
- Design and Algorithms
- Implementation with code
- Results
- Conclusion

## Introduction

Decentralized finance (DeFi), an emerging financial technology that aims to remove intermediaries in financial transactions, has opened up multiple avenues of income for investors. Yield farming is one such investment strategy in DeFi. It involves lending or staking your cryptocurrency coins or tokens to get rewards in the form of transaction fees or interest. This is somewhat similar to earning interest from a bank account; you are technically lending money to the bank. Only yield farming can be riskier, volatile, and complicated unlike putting money in a bank.

Yield farming involves moving crypto through different marketplaces. There is also an element of yield farming where the strategy becomes less effective when more people know about it. But yield farming is currently the most significant growth driver of the DeFi sector, helping it expand from a market cap of \$500 million to \$10 billion in 2020 alone. Here's a primer on yield farming.

**Yield farming** is essentially the practice of token holders finding ways of using their assets to earn returns. Depending on how the assets are utilized, the returns may take different forms. For example, by serving as liquidity providers in Uniswap, a 'farmer' can earn returns in the form of a share of the trading fees every time some agent swaps tokens. Alternatively, depositing the tokens in Compound earns interest, as these tokens are lent out to a borrower who pays interest

## Problem Statement

Develop a cryptocurrency based decentralized application using yield farming

### Explanation

The initial stage in yield farming is to deposit the money into a liquidity pool, which is effectively a smart contract containing funds. These pools fuel a marketplace where users may trade, borrow or lend tokens. You've officially become a liquidity provider after you've added your cash to a pool.

You'll be rewarded with revenue produced by the underlying DeFi technology in exchange for locking up your findings in the pool. It is important to note that investing in ETH does not qualify as yield farming. Instead, yield farming is the practice of lending out ETH on a decentralized non-custodial money market protocol like Aave and then collecting a return. Reward tokens may also be put in liquidity pools, and its normal practice for users to move their assets across protocols in order to get greater returns.

It's complicated. Yield farmers are frequently highly familiar with the Ethereum network and its complexities, and they will transfer their assets around to multiple DeFi platforms to maximize their profits. It's not easy, and it's certainly not easy money. Those who supply liquidity are likewise paid based on the quantity of liquidity given, so those who reap massive benefits have equally massive sums of money behind

---

## Design and Algorithm

A DeFi yield farming dApp presents a platform to stake the farmer's coins and moreover also facilitate the automation of reward payments for the liquidity provider.

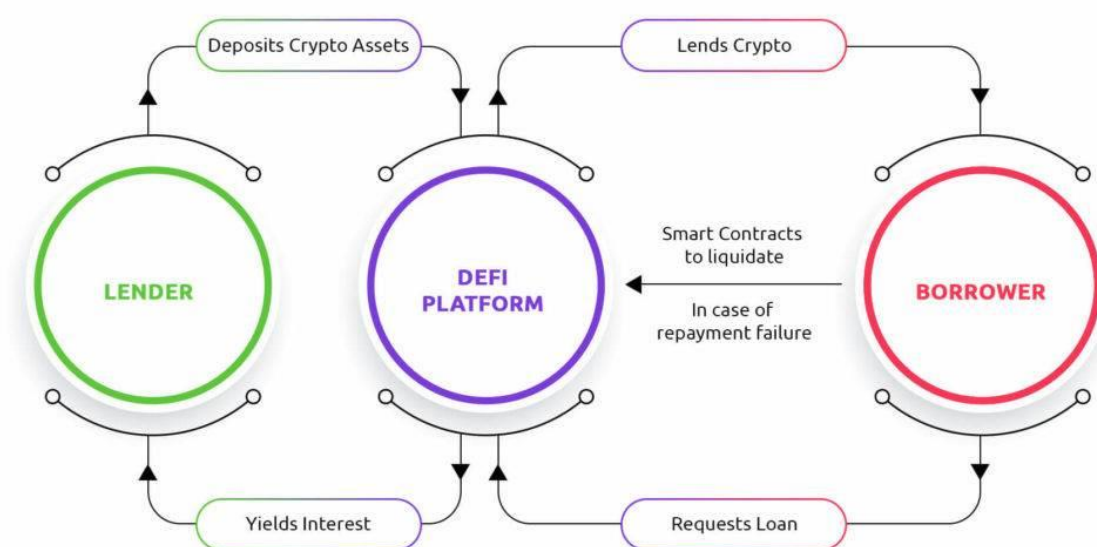
A decentralized app is an application that works on a decentralized network. It uses backend smart contracts for app logic and the blockchain for data storage. Since they are decentralized, they are not owned by an individual or a company and can't be taken down. As is with all blockchain technology it is transparent and automated through the binding of the social contract. The cryptography backbone makes the dApp secure against fraud. Overall presenting a good solution to store transfer and receive tokens.

The strategies involved in yield farming are based on at least one of these: lending, borrowing, supplying capital to liquidating pools and staking liquidity provider tokens.

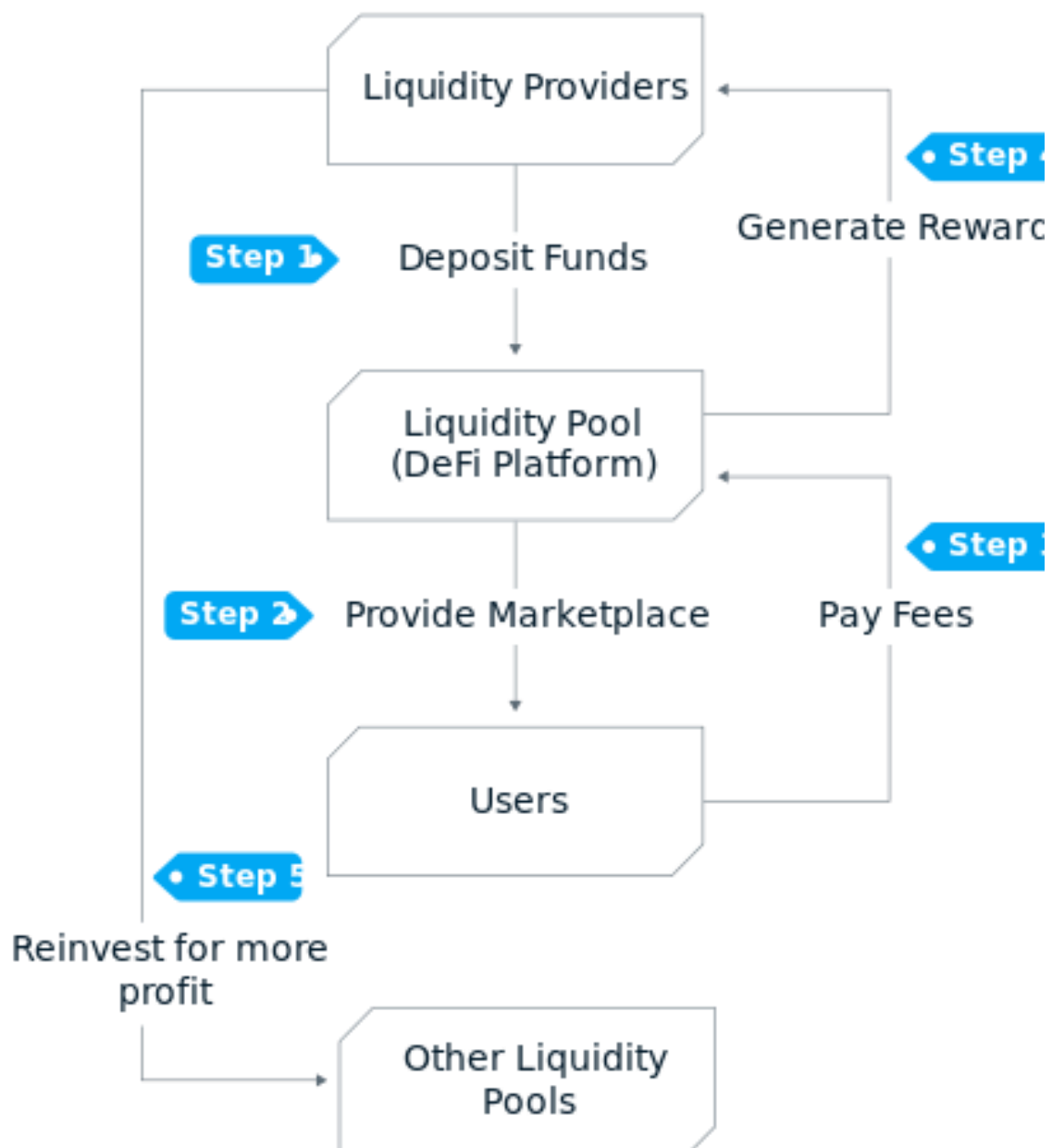
- **Lending:** By lending tokens to DeFi platforms, the farmers make a return on the tokens they've locked with the platform as it gives them extra coins. These tokens then through liquidity mining can be exchanged or reinvested to gain the returns.
- **Borrowing:** On the other hand, borrowing tokens allows the farmer to use them as collateral in a different protocol. Swapping the coins to different protocols and repeating this cycle also generates a high reward based on the initial capital.

- **Supplying coins to Liquidity pools:** Liquidity pools are smart contracts with tokens deposited to provide liquidity to the DeFi platform. Each pool has a pair of tokens to trade. When the pool contract is created the balance is set at zero tokens. So, the initial provider, i.e., the first person investing, set the price of the pool. The value of both tokens must be equal to reduce the risk of arbitrage (an opportunity for external sources to get the tokens at a low price and reinvest immediately to another platform). The providers that follow must invest in both tokens proportionally to avoid the same arbitrage risk. For Example, in Uniswap a pair of ERC 20 tokens are traded. The return for the pool is given in the form of a liquidity token, these are tradeable assets that can be swapped or sold.

### Architecture Workflow



## Algorithm Workflow



## Implementation

### Decentral Bank. Sol

```
1 pragma solidity ^0.5.0;
2
3 import './RWD.sol';
4 import './Tether.sol';
5
6 contract DecentralBank {
7     string public name = 'Decentral Bank';
8     address public owner;
9     Tether public tether;
10    RWD public rwd;
11
12    address[] public stakers;
13
14    mapping(address => uint) public stakingBalance;
15    mapping(address => bool) public hasStaked;
16    mapping(address => bool) public isStaking;
17
18    constructor(RWD _rwd, Tether _tether) public {
19        rwd = _rwd;
20        tether = _tether;
21        owner = msg.sender;
22    }
23
24    // staking function
25    function depositTokens(uint _amount) public {
26
27        // require staking amount to be greater than zero
28        require(_amount > 0, 'amount cannot be 0');
29
30        // Transfer tether tokens to this contract address for staking
31        tether.transferFrom(msg.sender, address(this), _amount);
32
33        // Update Staking Balance
34        stakingBalance[msg.sender] = stakingBalance[msg.sender] + _amount;
35
36        if(!hasStaked[msg.sender]) {
37            stakers.push(msg.sender);
38        }
39
40        // Update Staking Balance
41        isStaking[msg.sender] = true;
42        hasStaked[msg.sender] = true;
43    }
44
45    // unstake tokens
46    function unstakeTokens() public {
47        uint balance = stakingBalance[msg.sender];
48        // require the amount to be greater than zero
49        require(balance > 0, 'staking balance cannot be less than zero');
50
51        // transfer the tokens to the specified contract address from our bank
52        tether.transfer(msg.sender, balance);
53
54        // reset staking balance
55        stakingBalance[msg.sender] = 0;
56
57        // Update Staking Status
58        isStaking[msg.sender] = false;
59    }
60
61    // issue rewards
62    function issueTokens() public {
63        // Only owner can call this function
64        require(msg.sender == owner, 'caller must be the owner');
65
66        // issue tokens to all stakers
67        for (uint i=0; i<stakers.length; i++) {
68            address recipient = stakers[i];
69            uint balance = stakingBalance[recipient] / 9;
70            if(balance > 0) {
71                rwd.transfer(recipient, balance);
72            }
73        }
74    }
75
76 }
77 }
```



## Migrations. Sol

```
1 pragma solidity ^0.5.0;
2
3 contract Migrations {
4     address public owner;
5     uint public last_completed_migration;
6
7     constructor() public {
8         owner = msg.sender;
9     }
10
11     modifier restricted() {
12         if (msg.sender == owner) _;
13     }
14
15     function setCompleted(uint completed) public restricted {
16         last_completed_migration = completed;
17     }
18
19     function upgrade(address new_address) public restricted {
20         Migrations upgraded = Migrations(new_address);
21         upgraded.setCompleted(last_completed_migration);
22     }
23 }
```

## Tethers. Sol

```
1 pragma solidity ^0.5.0;
2
3 contract Tether {
4     string public name = "Mock Tether Token";
5     string public symbol = "mUSDT";
6     uint256 public totalSupply = 1000000000000000000000000; // 1 million tokens
7     uint8 public decimals = 18;
8
9     event Transfer(
10         address indexed _from,
11         address indexed _to,
12         uint _value
13     );
14
15     event Approval(
16         address indexed _owner,
17         address indexed _spender,
18         uint _value
19     );
20
21     mapping(address => uint256) public balanceOf;
22     mapping(address => mapping(address => uint256)) public allowance;
23
24     constructor() public {
25         balanceOf[msg.sender] = totalSupply;
26     }
27
28     function transfer(address _to, uint256 _value) public returns (bool success) {
29         // require that the value is greater or equal for transfer
30         require(balanceOf[msg.sender] >= _value);
31         // transfer the amount and subtract the balance
32         balanceOf[msg.sender] -= _value;
33         // add the balance
34         balanceOf[_to] += _value;
35         emit Transfer(msg.sender, _to, _value);
36         return true;
37     }
38
39     function approve(address _spender, uint256 _value) public returns (bool success) {
40         allowance[msg.sender][_spender] = _value;
41         emit Approval(msg.sender, _spender, _value);
42         return true;
43     }
44
45     function transferFrom(address _from, address _to, uint256 _value) public returns
46     (bool success) {
47         require(_value <= balanceOf[_from]);
48         require(_value <= allowance[_from][msg.sender]);
49         // add the balance for transferFrom
50         balanceOf[_to] += _value;
51         // subtract the balance for transferFrom
52         balanceOf[_from] -= _value;
53         allowance[msg.sender][_from] -= _value;
54         emit Transfer(_from, _to, _value);
55         return true;
56     }
57 }
```



```

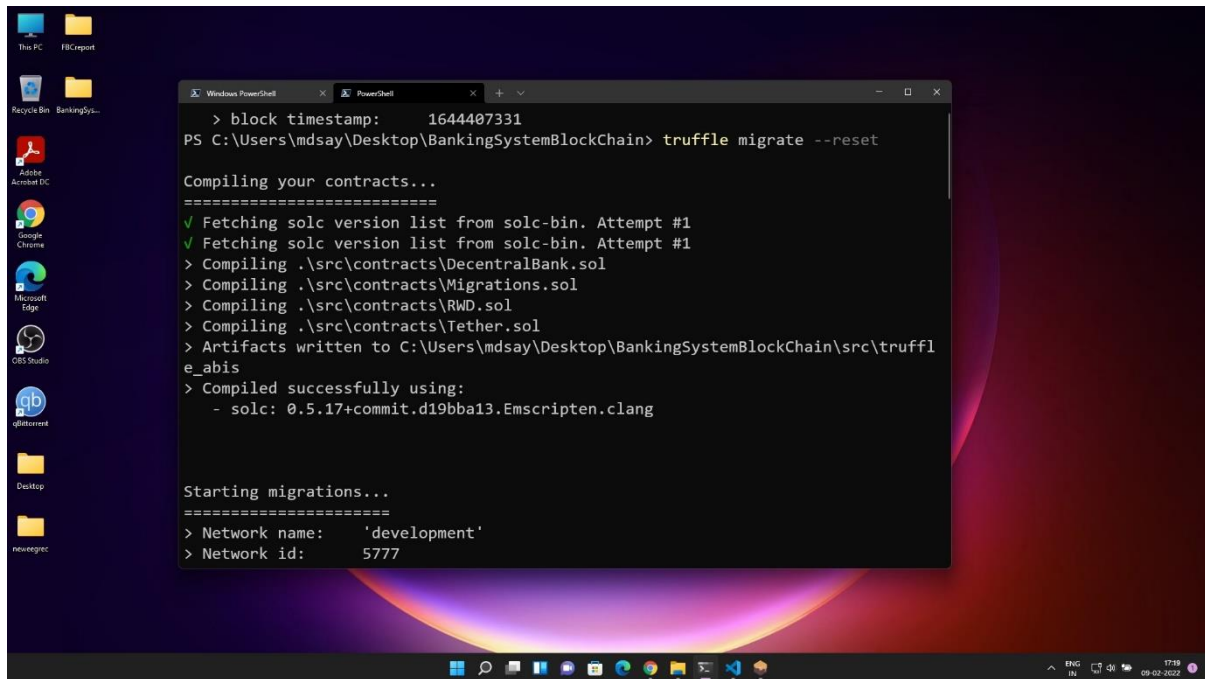
1 import React, { Component } from "react";
2 import Navbar from "../Navbar";
3 import Web3 from "web3";
4 import "../App.css";
5 import Main from "../Main";
6 import Tether from "../truffle_abi/Tether.json";
7 import RWD from "../truffle_abi/RWD.json";
8 import DecentralBank from "../truffle_abi/DecentralBank.json";
9 import ParticleSettings from "../ParticleSettings";
10
11 class App extends Component {
12   async componentWillMount() {
13     await this.loadWeb3();
14     await this.loadBlockchainData();
15   }
16
17   async loadBlockchainData() {
18     const web3 = window.web3;
19     const accounts = await web3.eth.getAccounts();
20     console.log(accounts);
21     this.setState({ account: accounts[0] });
22     const networkId = await web3.eth.net.getId();
23
24     //LOAD Tether TOKEN
25     const tetherData = Tether.networks[networkId];
26     if (tetherData) {
27       const tether = new web3.eth.Contract(Tether.abi,
28 tetherData.address);
29       let tetherBalance = await tether.methods
30         .balanceOf(this.state.account)
31         .call();
32       this.setState({ tetherBalance: tetherBalance.toString() });
33     } else {
34       window.alert("tether contract not deployed to detect network");
35     }
36
37     //LOAD RWD TOKEN
38     const rwdTokenData = RWD.networks[networkId];
39     if (rwdTokenData) {
40       const rwd = new web3.eth.Contract(RWD.abi, rwdTokenData.address);
41       this.setState({ rwd });
42       let rwdTokenBalance = await rwd.methods
43         .balanceOf(this.state.account)
44         .call();
45       this.setState({ rwdTokenBalance: rwdTokenBalance.toString() });
46     } else {
47       window.alert("Reward Token contract not deployed to detect network");
48     }
49
50     //Load DecentralBank
51     const decentralBankData = DecentralBank.networks[networkId];
52     if (decentralBankData) {
53       const decentralBank = new web3.eth.Contract(
54         DecentralBank.abi,
55         decentralBankData.address
56       );
57       this.setState({ decentralBank });
58       let stakingBalance = await decentralBank.methods
59         .stakingBalance(this.state.account)
60         .call();
61       this.setState({ stakingBalance: stakingBalance.toString() });
62     } else {
63       window.alert("TokenForm contract not deployed to detect network");
64     }
65
66     this.setState({ loading: false });
67   }
68
69   async loadWeb3() {
70     if (window.ethereum) {
71       window.web3 = new Web3(window.ethereum);
72       await window.ethereum.enable();
73     } else if (window.web3) {
74       window.web3 = new Web3(window.web3.currentProvider);
75     } else {
76       window.alert(
77         "Non ethereum browser detected. You should consider Metamask!"
78       );
79     }
80   }
81
82   stakeTokens = (amount) => {
83     this.setState({ loading: true });
84     this.state.tether.methods
85       .approve(this.state.decentralBank._address, amount)
86       .send({ from: this.state.account })
87       .on("transactionHash", (hash) => {
88         this.state.decentralBank.methods
89           .depositTokens(amount)
90           .send({ from: this.state.account })
91           .on("transactionHash", (hash) => {
92             this.setState({ loading: false });
93           });
94       });
95   };
96
97   unstakeTokens = () => {
98     this.setState({ loading: true });
99     this.state.decentralBank.methods
100       .unstakeTokens()
101       .send({ from: this.state.account })
102       .on("transactionHash", (hash) => {
103         this.setState({ loading: false });
104       });
105   };
106
107 ...

```

```
1 ...
2 constructor(props) {
3   super(props);
4   this.state = {
5     account: "0x0",
6     tether: {},
7     rwd: {},
8     decentralBank: {},
9     tetherBalance: "0",
10    rwdTokenBalance: "0",
11    stakingBalance: "0",
12    loading: true,
13  };
14 }
15
16 render() {
17   let content;
18
19   {
20     this.state.loading
21     ? (content = (
22       <p
23         id="loader"
24         className="text-center"
25         style={{ color: "white", margin: "30px" }}
26       >
27         LOADING PLEASE...
28       </p>
29     ))
30     : (content = (
31       <Main
32         tetherBalance={this.state.tetherBalance}
33         rwdBalance={this.state.rwdTokenBalance}
34         stakingBalance={this.state.stakingBalance}
35         stakeTokens={this.stakeTokens}
36         unstakeTokens={this.unstakeTokens}
37         decentralBankContract={this.decentralBank}
38       />
39     ));
40   }
41
42   return (
43     <div className="App" style={{ position: "relative" }}>
44       <div style={{ position: "absolute" }}>
45         <ParticleSettings />
46       </div>
47       <Navbar account={this.state.account} />
48       <div className="container-fluid mt-5">
49         <div className="row">
50           <main
51             role="main"
52             className="col-lg-12 ml-auto mr-auto"
53             style={{ maxWidth: "600px" }}
54             // eslint-disable-next-line react/jsx-no-duplicate-props
55             style={{ minHeight: "100vm" }}
56           >
57             <div>{content}</div>
58           </main>
59         </div>
60       </div>
61     </div>
62   );
63 }
64 }
65
66 export default App;
```

## Results

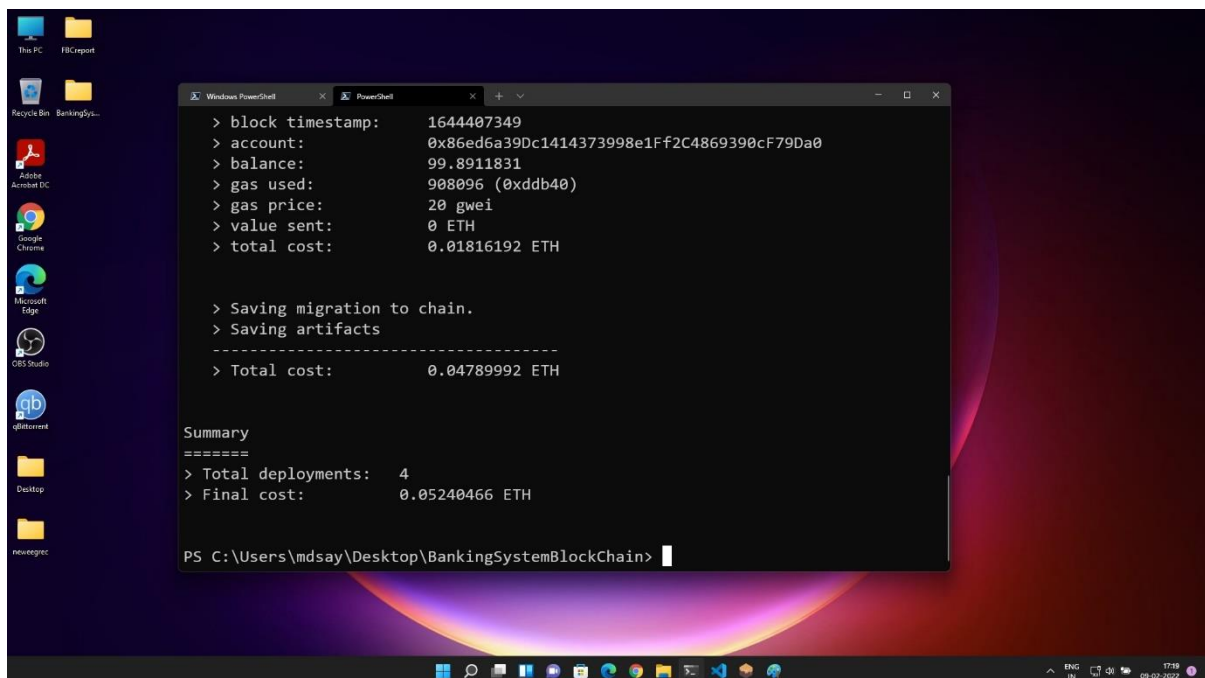
### Deploying contracts



```
> block timestamp: 1644407331
PS C:\Users\mdsay\Desktop\BankingSystemBlockchain> truffle migrate --reset

Compiling your contracts...
=====
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Fetching solc version list from solc-bin. Attempt #1
> Compiling .\src\contracts\DecentralBank.sol
> Compiling .\src\contracts\Migrations.sol
> Compiling .\src\contracts\RWD.sol
> Compiling .\src\contracts\Tether.sol
> Artifacts written to C:\Users\mdsay\Desktop\BankingSystemBlockchain\src\truffle_abis
> Compiled successfully using:
   - solc: 0.5.17+commit.d19bba13.Emscripten.clang

Starting migrations...
=====
> Network name: 'development'
> Network id: 5777
```



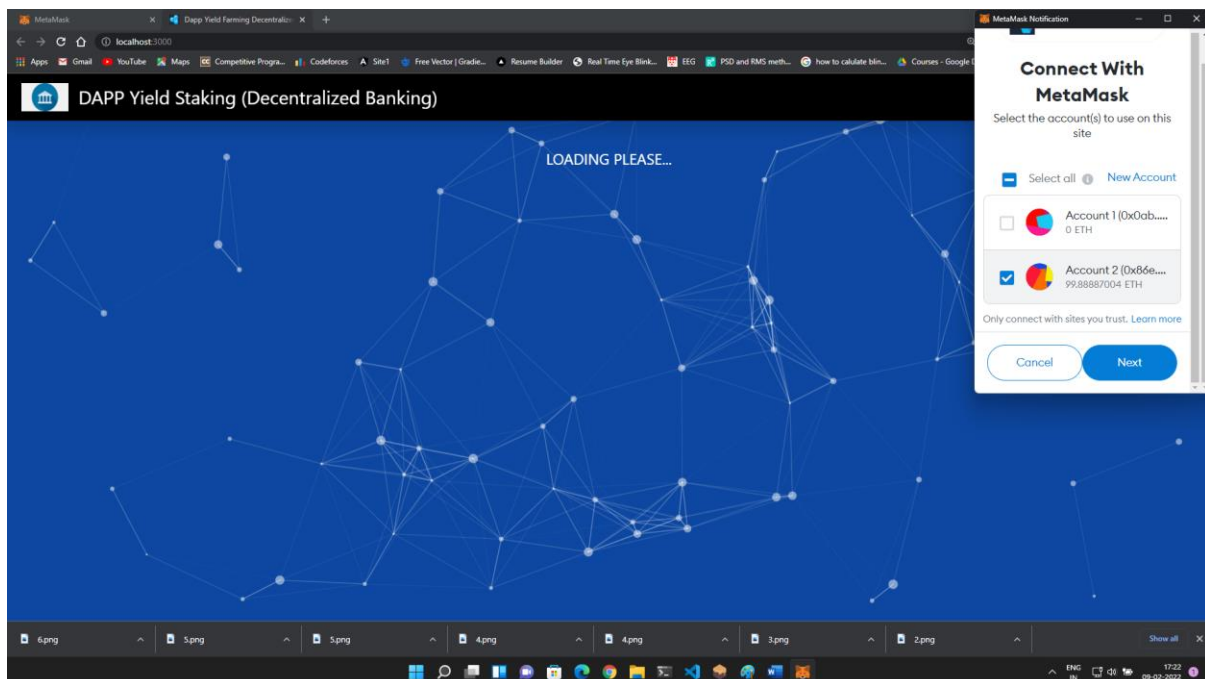
```
> block timestamp: 1644407349
> account: 0x86ed6a39Dc1414373998e1Ff2C4869390cF79Da0
> balance: 99.8911831
> gas used: 908096 (0xddb40)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.01816192 ETH

> Saving migration to chain.
> Saving artifacts
=====
> Total cost: 0.04789992 ETH

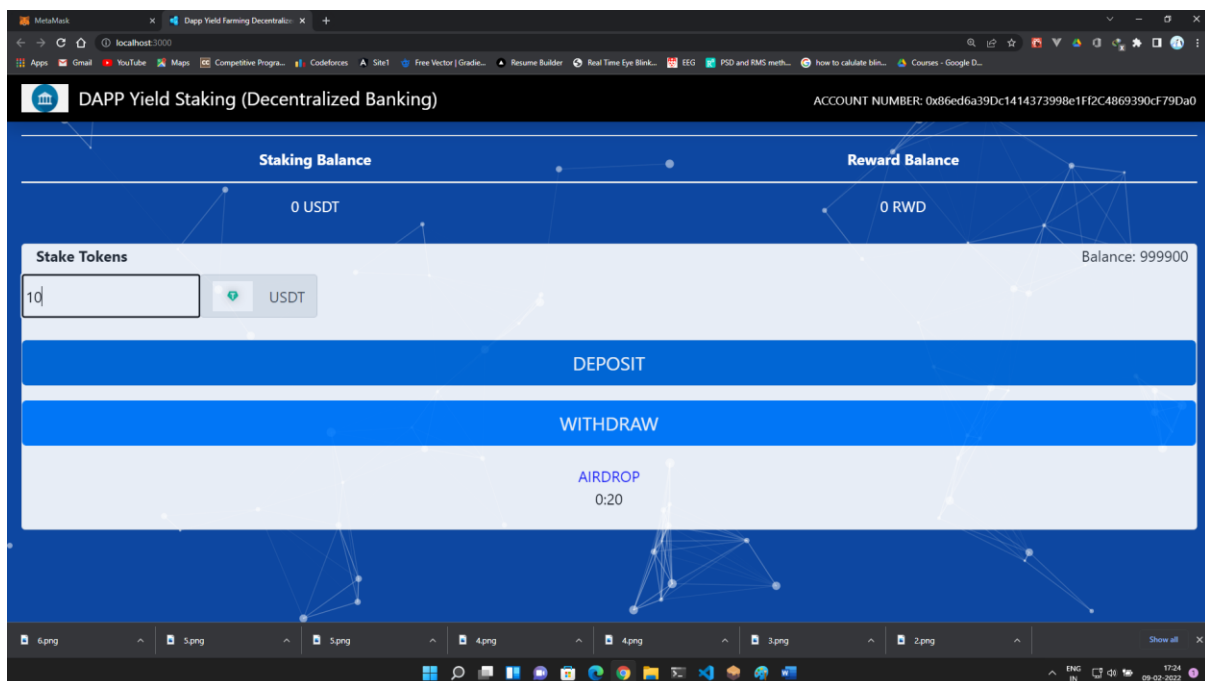
Summary
=====
> Total deployments: 4
> Final cost: 0.05240466 ETH

PS C:\Users\mdsay\Desktop\BankingSystemBlockchain>
```

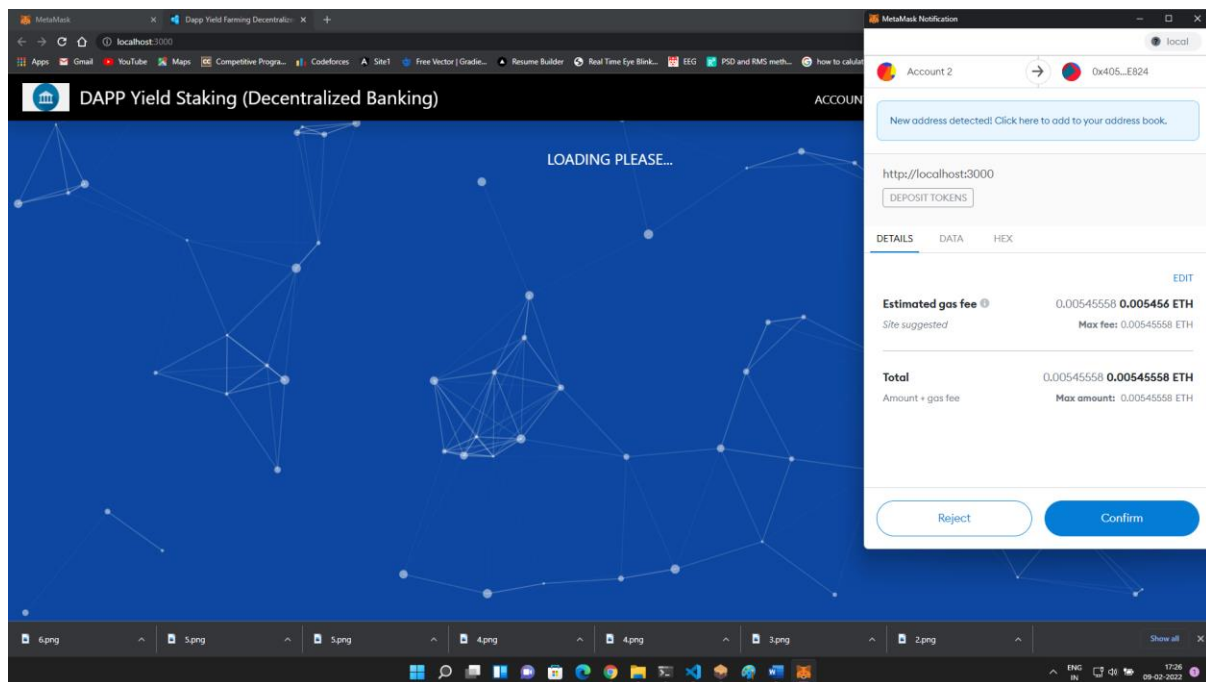
## Ping Server and Connect Meta mask Account



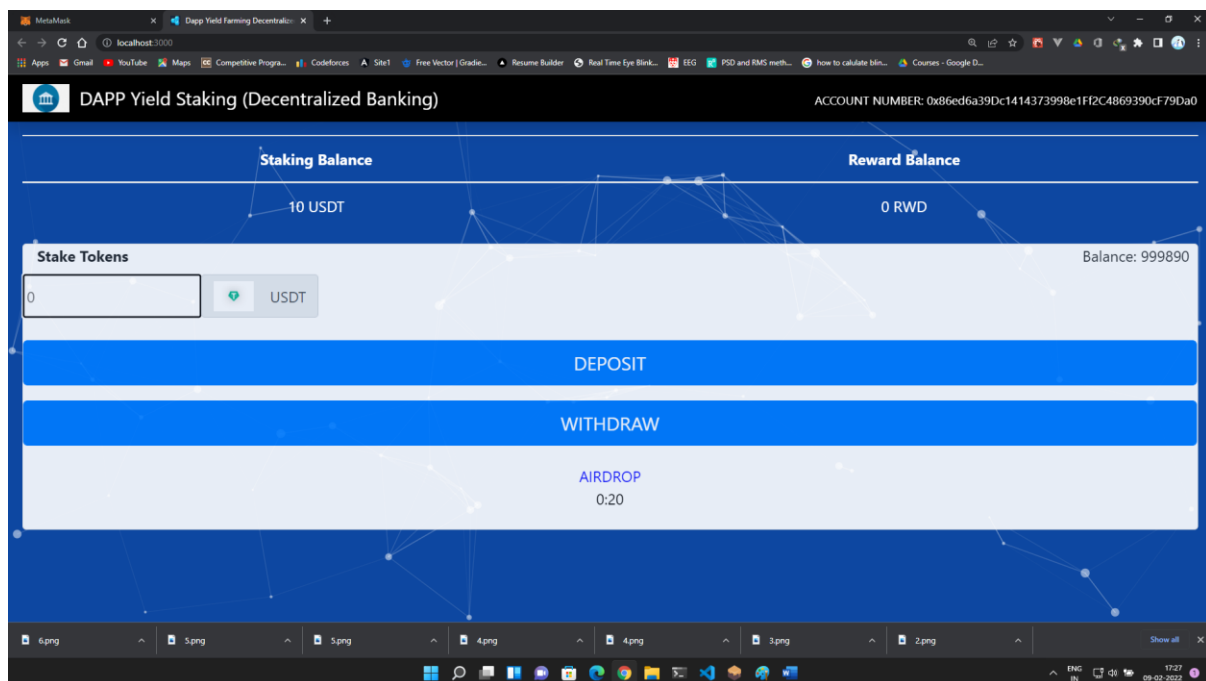
## Deposit 10 USDT Tokens



## Approve Transaction

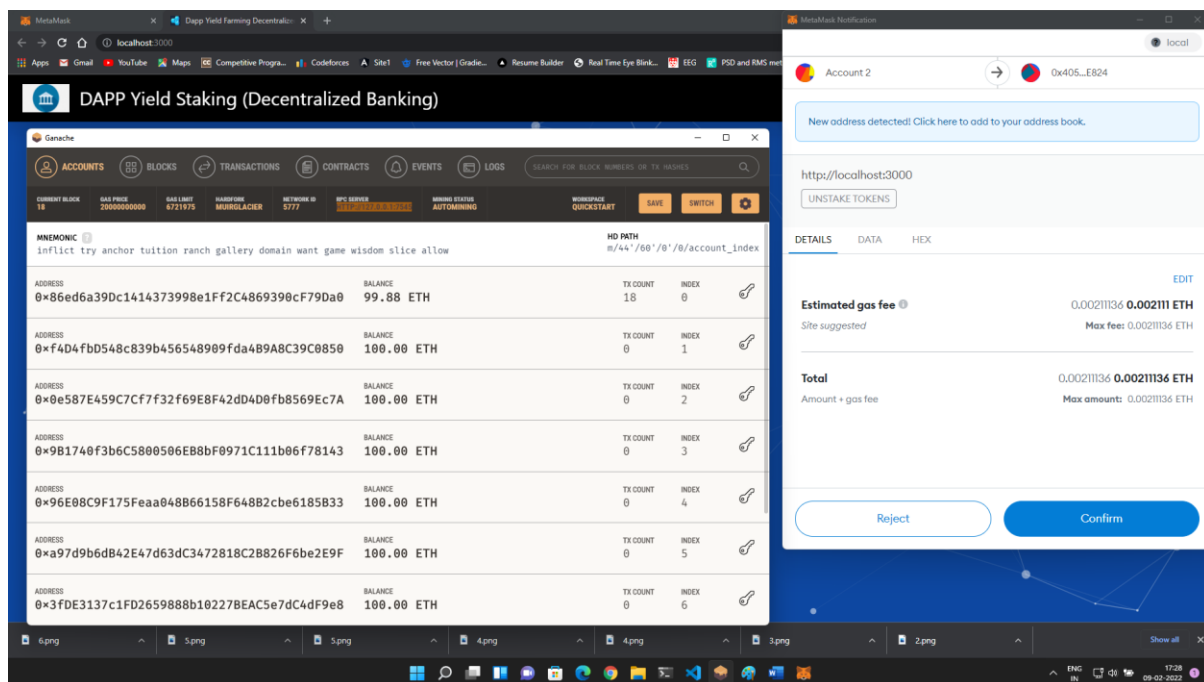


## Updated Decentral Bank Balance

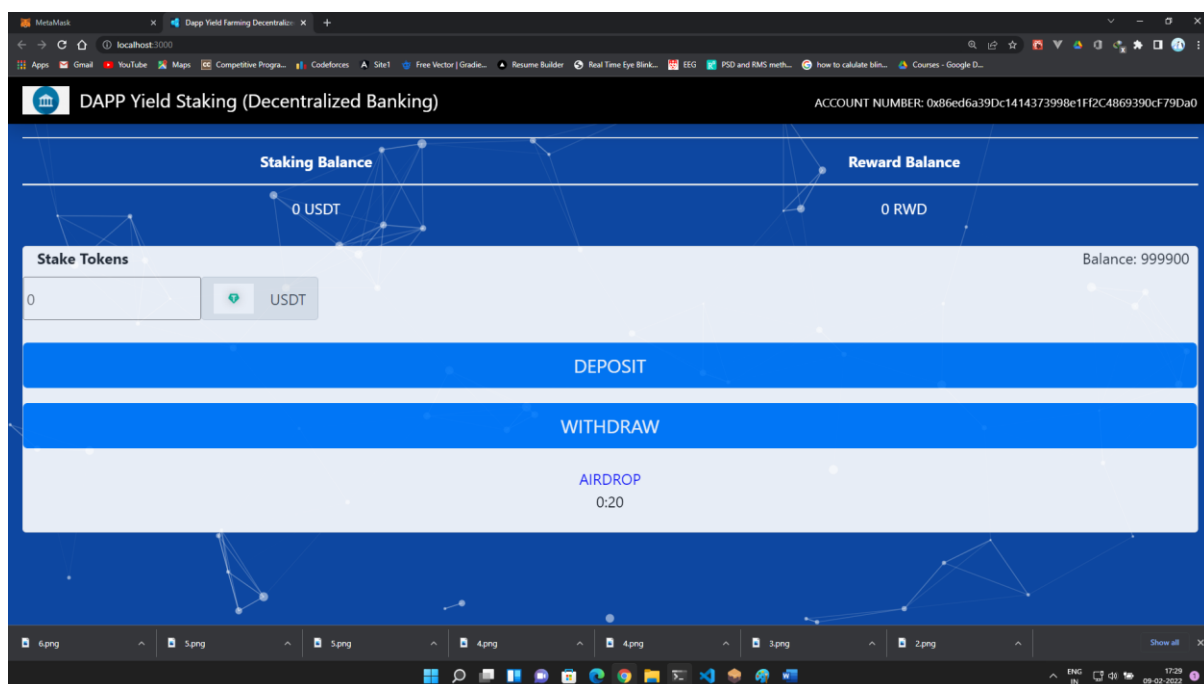




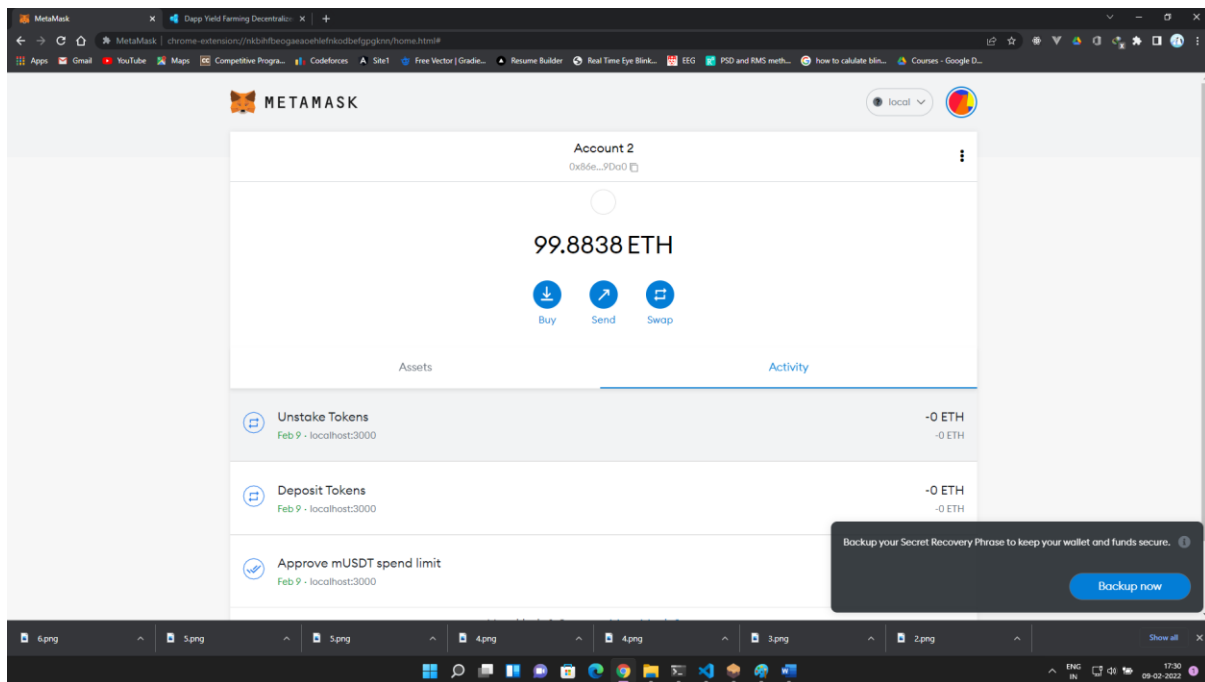
## Withdraw Balance



## Withdrawn and Updated Balance



## Metamask Activity log



## Conclusion

- Decentralized Application was built using Meta mask, Web3.0, Ganache and React
- User was able to authenticate, authorize crypto wallet and performed transaction
- User was able to Stake, Store and Withdraw tokens from the bank