

PROJECT REPORT

CHAIN CONNECT NODES

[DISTRIBUTED LEDGER BLOCKCHAIN]

1. **INTRODUCTION**
 - 1.1 Project Overview
 - 1.2 Purpose
2. **LITERATURE SURVEY**
 - 2.1 Existing problem
 - 2.2 References
 - 2.3 Problem Statement Definition
3. **IDEATION & PROPOSED SOLUTION**
 - 3.1 Empathy Map Canvas
 - 3.2 Ideation & Brainstorming
4. **REQUIREMENT ANALYSIS**
 - 4.1 Functional requirement
 - 4.2 Non-Functional requirements
5. **PROJECT DESIGN**
 - 5.1 Data Flow Diagrams & User Stories
 - 5.2 Solution Architecture
6. **PROJECT PLANNING & SCHEDULING**
 - 6.1 Technical Architecture
 - 6.2 Sprint Planning & Estimation
 - 6.3 Sprint Delivery Schedule
7. **CODING & SOLUTIONING (Explain the features added in the project along with code)**
 - 7.1 Key Features
8. **PERFORMANCE TESTING**
 - 8.1 Performace Metrics
9. **RESULTS**
 - 9.1 Output Screenshots
10. **ADVANTAGES & DISADVANTAGES**
11. **CONCLUSION**
12. **FUTURE SCOPE**
13. **APPENDIX**

Source Code, GitHub & Project Demo Link

CHAIN CONNECT NODES

[DISTRIBUTED LEDGER BLOCKCHAIN]

1. INTRODUCTION:

1.1 PROJECT OVERVIEW:

The project “**CHAIN CONNECT NODES - DISTRIBUTED LEDGER BLOCKCHAIN**” is focused on developing a decentralized application that facilitates secure and timestamped financial transactions between senders and receivers. It employs a tech stack that includes Solidity for smart contract development, Node.js for server-side scripting, MetaMask for wallet integration, and ChainConnect nodes for Ethereum network connectivity. Key features of the application include a robust Solidity smart contract, a Node.js server that interacts with the blockchain and offers an API for transaction management, a user-friendly web interface with MetaMask integration for wallet management, and reliable timestamping of transactions to ensure data integrity. Security measures are paramount, with testing on Ethereum test networks preceding deployment on the mainnet. Clear documentation and user education will guide users, and ongoing maintenance and compliance with legal and regulatory standards are integral to the project's success. This project aims to provide a secure, transparent, and user-friendly solution for financial transactions, harnessing the power of blockchain technology to ensure trust and data immutability.

1.1 PURPOSE:

The purpose of this project is to create a secure, transparent, and decentralized system for financial transactions between senders and receivers using blockchain technology. The key objectives and purposes of this project include:

- 1. Security:** Enhancing the security of financial transactions by leveraging blockchain's cryptographic and consensus mechanisms to protect against fraud and unauthorized access.
- 2. Transparency:** Providing a transparent and immutable ledger of all financial transactions, making it easy to verify and audit transaction history.
- 3. Trust:** Establishing a trustless environment where participants can transact directly with one another without relying on intermediaries, fostering trust in the system.

4. Timestamping: Recording transactions with timestamps to ensure the chronological order and accuracy of financial activities.

5. User-Friendly Interaction: Offering a user-friendly web interface with MetaMask integration to make it accessible to a wide range of users and simplify transaction management.

6. Compliance: Ensuring compliance with legal and regulatory requirements, especially in financial transactions, to operate within the boundaries of the law.

7. Blockchain Technology Utilization: Harnessing the potential of blockchain technology to create a decentralized, tamper-resistant, and efficient system for financial transactions.

8. Documentation and User Education: Providing clear documentation and guidance to users to ensure they understand how to use the application and manage their transactions securely.

9. Maintenance and Upkeep: Ensuring the application's ongoing functionality and security through monitoring, updates, and prompt bug fixes.

In summary, the purpose of this project is to empower individuals and organizations to conduct secure, transparent, and trustless financial transactions using blockchain technology, enhancing data integrity and user control while adhering to legal and regulatory standards.

2. LITERATURE SURVEY:

2.1 EXISTING SYSTEM:

The existing system is dependent on centralized financial institutions, such as banks and payment processors, acting as intermediaries for financial transactions. This setup often leads to transaction fees imposed on users and significant delays in settling transactions. Additionally, the lack of transparency is a notable drawback, as transaction history is not always readily accessible to users, requiring them to rely on statements from their financial institutions. This traditional system necessitates trust in these centralized intermediaries to process transactions securely and accurately,

introducing the element of counterparty risk. Moreover, accessibility to financial services might be limited in certain regions or for specific individuals, which can hinder financial inclusion. Furthermore, traditional financial institutions are subject to strict compliance and regulation, including adherence to anti-money laundering (AML) and know your customer (KYC) rules, making the process complex and cumbersome. Lastly, the existing system lacks the inherent benefits of decentralization and security that are integral to blockchain technology.

2.1 REFERENCES:

- [1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System". <https://bitcoin.org/bitcoin.pdf>
- [2] Sundaram B.B., Wako M.D.A., Pandey A., Genale M.A.S., Janga M.V., Karthika P. - Supply chain management finance application in bank official website using blockchain 2022 6th International Conference on Intelligent Computing and Control Systems, ICICCS, IEEE (2022), pp. 812-817
- [3] Trollman, H.; Garcia-Garcia, G.; Jagtap, S.; Trollman, F. Blockchain for Ecologically Embedded Coffee Supply Chains. *Logistics*. 2022, 6, 43. [CrossRef]
- [4] Zhang T., Li J., Jiang X. - Analysis of supply chain finance based on blockchain *Procedia Comput. Sci.*, 187 (2021), pp. 1-6
- [5] Chen J., Cai T., He W., Chen L., Zhao G., Zou W., Guo L. - A blockchain-driven supply chain finance application for auto retail industry *Entropy*, 22 (1) (2020), p. 95
- [6] Y. Wang, J. H. Han and P. Beynon-Davies, "Understanding blockchain technology for future supply chains: a systematic literature review and research agenda", *Supply Chain Management: An International Journal*, 2019, [online] Available: http://orca.cf.ac.uk/115569/1/_system_appendPDF_proof_hi.pdf.
- [7] J. A. Tarr, "Distributed ledger technology blockchain and insurance: Opportunities risks and challenges", *Insurance Law Journal*, vol. 29, no. 3, pp. 254-268, 2018.

- [8] A Gupta , S Gupta - Blockchain technology application in Indian Banking Sector
Delhi Business Review , volume 19 , issue 2 , p. 75 - 84 Posted: 2018
- [9] A. Deshpande, K. Stewart, L. Lepetit and S. Gunashekar, "Distributed Ledger Technologies/Blockchain: Challenges opportunities and the prospects for standards", Overview report The British Standards Institution (BSI) 40, vol. 40, 2017, [online] Available:https://www.bsigroup.com/LocalFiles/zhtw/InfoSecnewsletter/No201706/download/BSI_Blockchain_DLT_Web.pdf.
- [10] D C Mills ,K Wang ,B Malone , A Ravi , J Marquardt ,A I Badev , . Ellithorpe M
- Distributed ledger technology in payments, clearing, and settlement
NMIMS Management Review , volume 36 , issue 2 , p. 60 - 89 Posted: 2016.

2.3 PROBLEM STATEMENT DEFINITION:

The current financial transaction landscape is plagued by multiple issues. Traditional systems impose significant transaction fees, causing users to incur substantial costs when sending and receiving funds. Additionally, transactions often suffer from delays, making it inconvenient for users who require quick access to their funds. The centralized nature of these systems also results in a lack of transparency, as users have limited access to real-time transaction records, forcing them to rely on periodic statements from financial institutions, which can hinder their ability to verify transactions as they occur.

Furthermore, there's a significant level of counterparty risk involved, as users must trust these centralized intermediaries to handle their transactions securely and accurately. This trust can sometimes be misplaced, leading to errors or fraud. Additionally, many individuals and regions lack access to financial services, limiting financial inclusion.

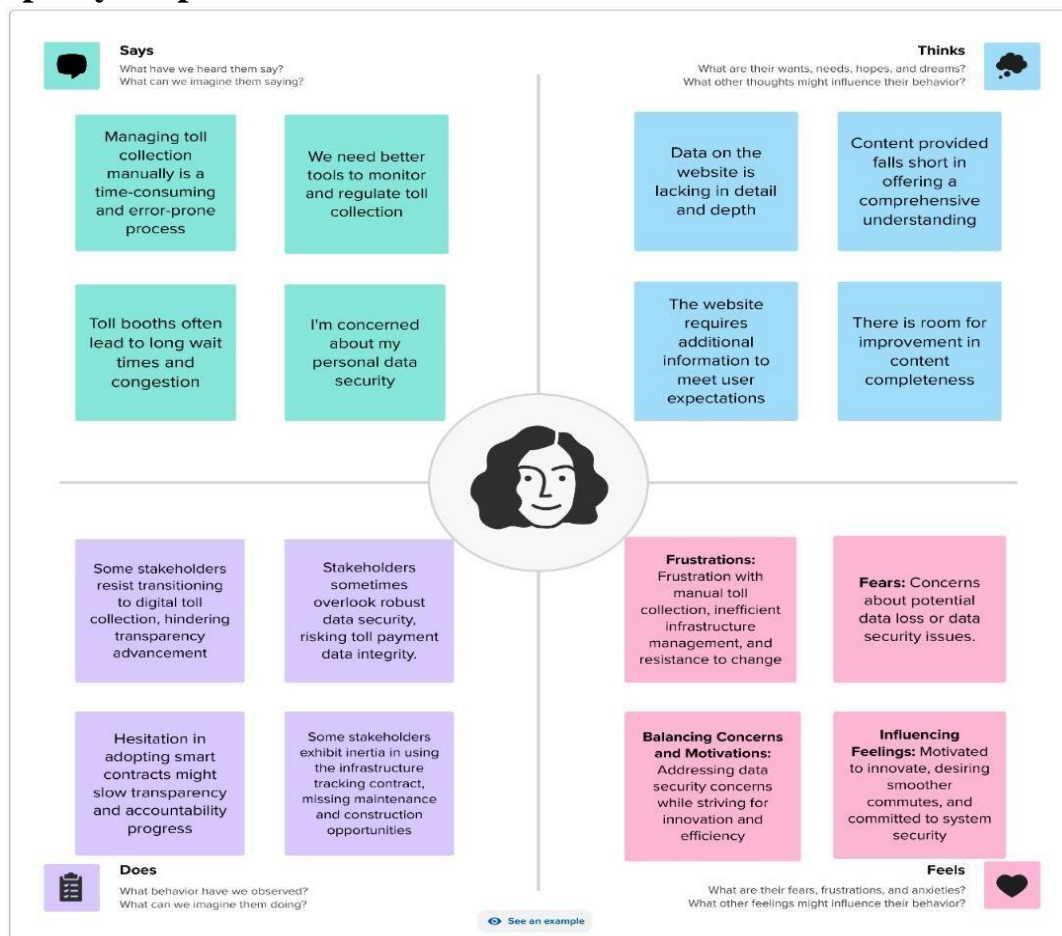
The complex regulatory landscape further exacerbates these problems, as financial institutions are required to adhere to various regulations, including anti-money laundering (AML) and know your customer (KYC) rules. This not only slows down transaction processing but also increases administrative overhead

Lastly, the centralized nature of the existing financial system increases the vulnerability to data breaches and system failures, posing risks to data security.

To address these extensive challenges and enhance the efficiency, security, and transparency of financial transactions, the proposed project will leverage blockchain technology, smart contracts, and decentralized infrastructure to create a more accessible, secure, and cost-effective financial transaction solution. This project will provide a trustless environment that eliminates the need for intermediaries, offering a decentralized ledger for transparent and timestamped transactions to mitigate the limitations of the existing financial system.


3. IDEATION & PROPOSED SOLUTION:

3.1 Empathy Map Canvas:



3.1 IDEATION & BRAINSTORMING:

Template



Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

🕒 10 minutes to prepare
🕒 1 hour to collaborate
👤 2-8 people recommended

➔

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 10 minutes

A

Team gathering

Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B

Set the goal

Think about the problem you'll be focusing on solving in the brainstorming session.

C

Learn how to use the facilitation tools

Use the Facilitation Superpowers to run a happy and productive session.

[Open article](#) ➔

1


Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

🕒 5 minutes


PROBLEM


How can we improve toll collection and infrastructure management for better efficiency and security, considering data security concerns and resistance to technology?





Key rules of brainstorming


To run an smooth and productive session


 Stay in topic.

 Encourage wild ideas.

 Defer judgment.

 Listen to others.

 Go for volume.

 If possible, be visual.

2 Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

TIP
You can select a sticky note and hit the small hand icon to move it around.

Person 1

Secure and efficient financial transactions, transparency, compliance.

Explore blockchain technology for financial operations.

Data security concerns, blockchain learning curve, need for user documentation.

Person 2

Convenient and intuitive financial transactions, low fees, simplicity.

Seamlessly manage day-to-day financial activities.

Complex interfaces, high transaction costs, lack of accessibility.

Person 3

Compliance solutions, regulatory adherence, data security.

Ensure financial platforms comply with evolving regulations.

Offer 24/7 customer support for immediate issue resolution.

Person 4

Cutting-edge blockchain features, exploration, and innovation.

Experiment with the latest blockchain technologies and applications.

Limited access to advanced features, lack of updates, and information on new blockchain developments.

3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

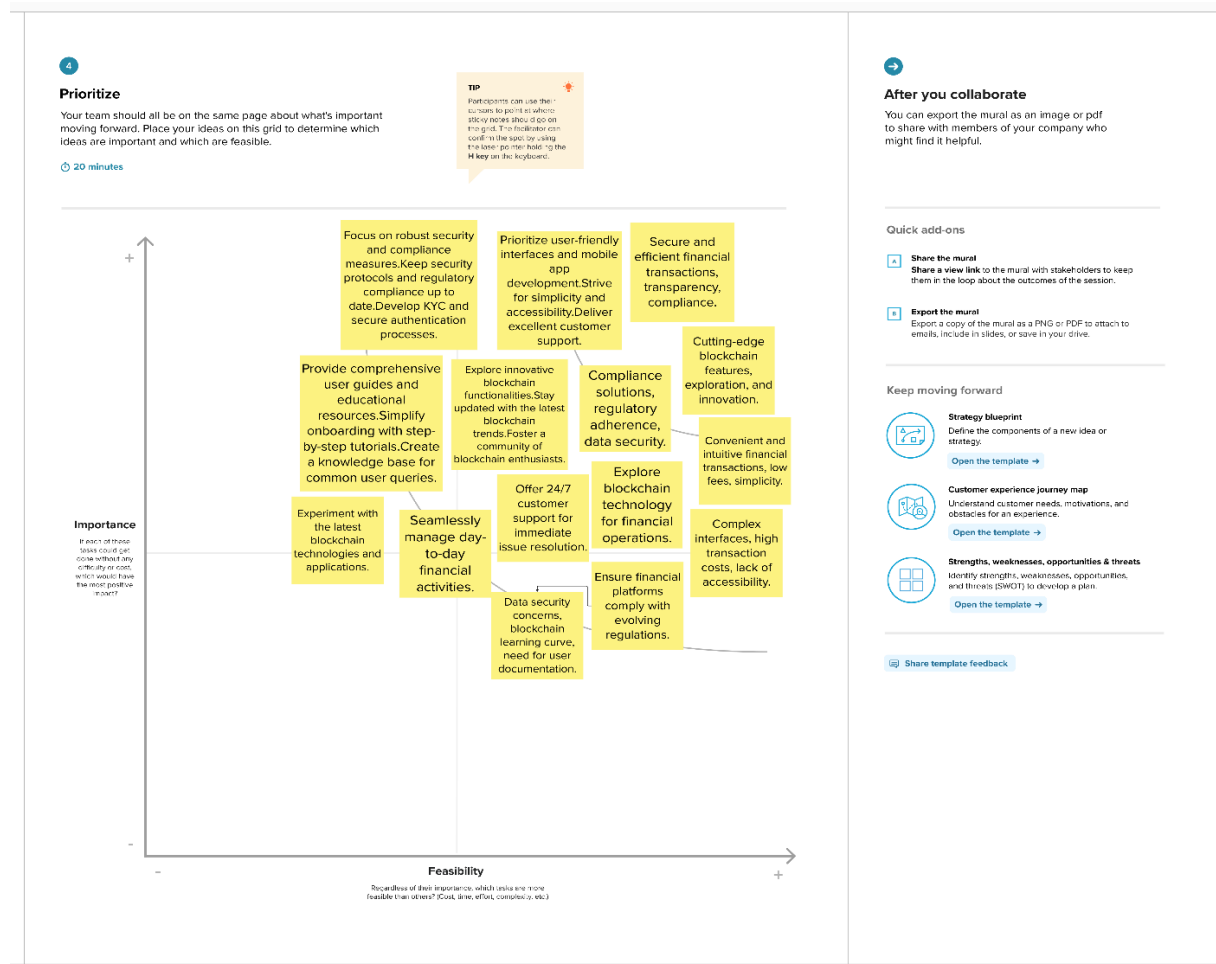
TIP
Add a sentence-like label to a sticky note to make it stand for the idea, theme, category, and category of your ideas. It helps others to find your ideas.

Focus on robust security and compliance measures. Keep security protocols and regulatory compliance up to date. Develop KYC and secure authentication processes.

Prioritize user-friendly interfaces and mobile app development. Strive for simplicity and accessibility. Deliver excellent customer support.

Explore innovative blockchain functionalities. Stay updated with the latest blockchain trends. Foster a community of blockchain enthusiasts.

Provide comprehensive user guides and educational resources. Simplify onboarding with step-by-step tutorials. Create a knowledge base for common user queries.



9. Security Measures: Robust security to protect accounts and data.
10. Testing Environment: A safe environment for practicing transactions.
11. Compliance and Regulation: User identity verification and legal compliance.
12. Documentation and User Guides: Clear instructions for users.
13. Notification System: Updates on transaction status.
14. User Support and Contact: A channel for user assistance.
15. Scalability: Design to accommodate growing user and transaction numbers.
16. Admin Dashboard: Tools for system administrators.
17. Audit Trail: Comprehensive logging of user activities and transactions.
18. Data Backup and Recovery: Safeguarding data and transaction history.

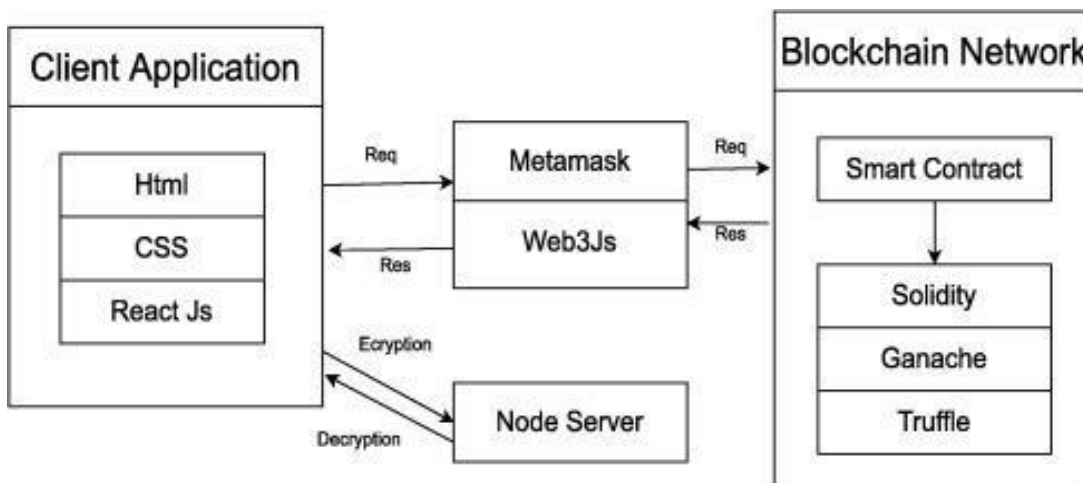
4.2 NON-FUNCTIONAL REQUIREMENT:

1. Performance: Efficient and responsive, even during peak usage.
2. Security: Robust security and data encryption.
3. Scalability: Easily accommodates growth.
4. Reliability: Minimal downtime and data loss prevention.
5. Usability: Intuitive user interface and clear instructions.
6. Compatibility: Works across browsers and devices.
7. Regulatory Compliance: Adheres to legal and financial regulations.
8. Auditability: Maintains comprehensive logs for auditing.
9. Data Backup and Recovery: Regular backups and recovery plan.
10. Response Time: Quick transaction and user action responses.
11. Availability: High availability with minimal downtime.
12. Documentation and Training: Comprehensive user and admin documentation.
13. Monitoring and Alerts: Monitors system performance and security.
14. Disaster Recovery: Plan for handling system failures.

15. Third-Party Integration: Supports third-party services.
16. Blockchain Standards: Adheres to industry blockchain standards.
17. Data Privacy: Safeguards user data and privacy compliance.and user satisfaction.

5. PROJECT DESIGN:

5.1 DATA FLOW DIAGRAM:

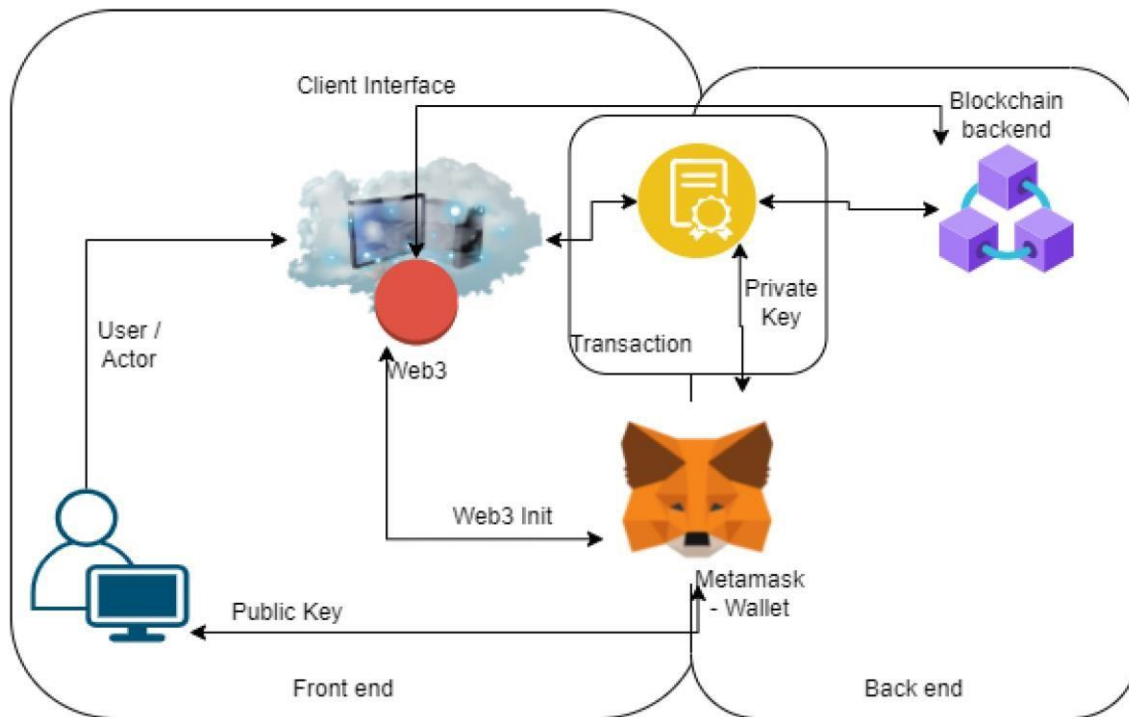


USER STORIES:

1. New users should be able to easily register accounts to get started with the platform. They expect a straightforward process for creating an account.
2. Users aim to connect their MetaMask wallets or other Ethereum wallets to the platform and ensure that their wallet addresses are accurately validated for a secure and smooth experience.
3. Senders and receivers both want the ability to initiate financial transactions on the platform by specifying the recipient's wallet address and the precise amount to be transferred.
4. Users prioritize secure transaction authorization and wish to securely sign and authorize transactions using their connected wallets, enhancing the overall security of their transactions.
5. Users value the transparency and verifiability of their transactions and thus expect to see timestamps on each transaction to track when they were completed.
6. Keeping track of their financial activities is crucial, so users look for a transaction history that provides them with a comprehensive list of past transactions, including sender and recipient details, transaction amounts, and timestamps.

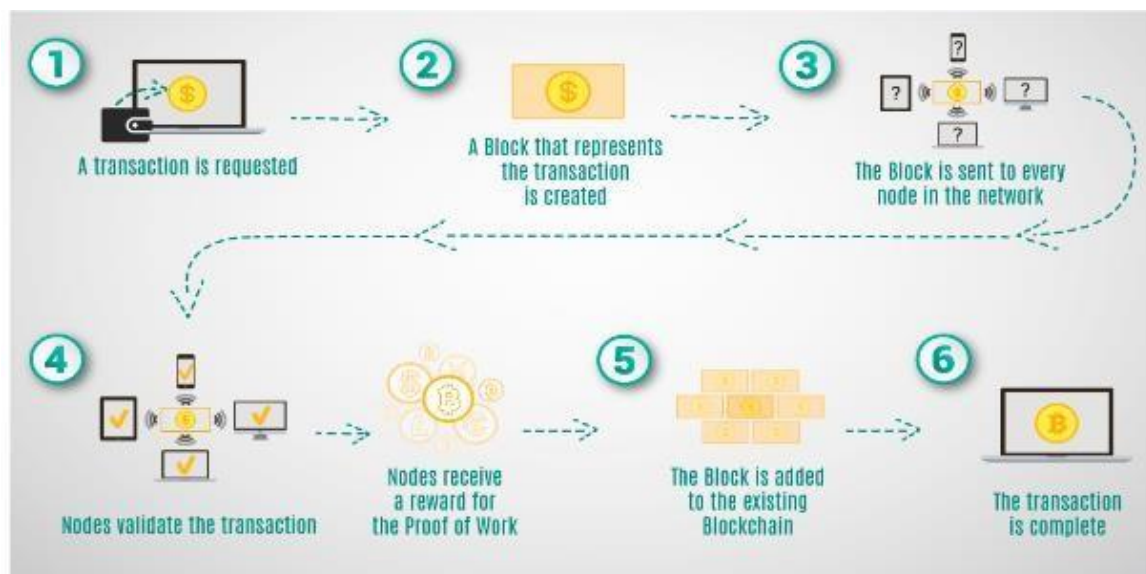
7. Security is a top priority, and users want assurance that their account and transaction data are safeguarded through robust security measures to instill trust in the platform.
8. New users desire a safe environment for practicing transactions without using real assets, allowing them to become comfortable with the platform.
9. Compliance with legal and regulatory requirements, such as Anti-Money Laundering (AML) and Know Your Customer (KYC) checks, is important, and users want a seamless process for verifying their identity.
10. Users also anticipate effective channels for contacting customer support to seek assistance or report issues when they encounter challenges while using the platform.
11. Keeping users informed about the status of their transactions is essential, and users request timely notifications and updates, including confirmations and critical alerts.
12. Administrators require a dashboard to monitor and manage user accounts, transactions, and system health, facilitating efficient administration.
13. For security and compliance purposes, administrators expect access to a comprehensive audit trail to review user activities and transactions.
14. Data security is paramount, and both users and administrators appreciate a robust data backup and recovery system to protect user data and transaction history.
15. Users and administrators alike value comprehensive documentation and guides explaining how to navigate the platform and manage accounts effectively.
16. Ensuring the protection and proper handling of personal data in compliance with data privacy regulations is crucial to user data privacy and security.

5.2 SOLUTION ARCHITECTURE:



6. PROJECT PLANNING:

6.1 TECHNICAL ARCHITECTURE:



6.2 SPRINT PLANNING & ESTIMATION:

- Estimating a project of this nature, a blockchain-based financial transaction application, involves several key considerations. First and foremost, the project duration is a critical factor, typically ranging from 6 months to a year, encompassing development, testing, and deployment phases. Assembling the right development team is essential, including blockchain developers, smart contract experts, full-stack developers, UI/UX designers, and security specialists, with the team's size varying based on project complexity. Development costs entail budgeting for salaries, infrastructure, and licenses, with the exact figures contingent on team rates and project duration. Infrastructure costs should also be factored in, encompassing blockchain network fees, such as gas fees, and server expenses, which include hosting, maintenance, and security.
- Robust testing and quality assurance processes require allocation of resources, varying based on project complexity. Additionally, planning for additional expenses is crucial, covering areas like documentation development, legal compliance consultation, project management, user training, contingency reserves (typically 10-20% of estimated costs for unforeseen issues), and post-launch maintenance and support. These estimations serve as a foundational framework and will necessitate further detailed breakdown as the project progresses, necessitating flexibility and adaptation based on real-world project developments.

6.3 SPRINT DELIVERY SCHEDULE

The delivery schedule for a blockchain-based financial transaction application can be summarized in several key phases. The project initiation phase, spanning the first two weeks, involves requirement gathering and team setup. Design and architecture, taking place in weeks three to six, encompass the creation of wireframes and system architecture. Smart contract development, occurring over weeks seven to ten, involves coding the transaction rules using Solidity. The frontend is developed in weeks eleven to fifteen, providing users with an interface for transaction initiation and wallet integration. The backend is developed in weeks sixteen to twenty, enabling communication between the frontend and the blockchain. A comprehensive testing and quality assurance phase takes place in weeks twenty-one to twenty-five to ensure robustness. Compliance and regulatory setup, if needed, occurs in weeks twenty-six

to twenty-eight, followed by user training and documentation development in weeks twenty-nine to thirty-one.

7. CODING AND SOLUTIONING:

Explanation: This project offers users the ability to securely initiate and verify transactions, manage wallet integration, and access comprehensive transaction history, all while adhering to compliance regulations and ensuring data security through robust security measures.

7.1 Key features :

a. Smart Contract:

- A Solidity smart contract will be developed to manage transactions, accounts, and timestamps.
- The contract will enforce secure and transparent financial transactions.

b. Node.js Server:

- A Node.js server will be created to interact with the Ethereum blockchain and the smart contract.
- It will provide an API for users to initiate and monitor transactions.

c. User Interface:

- A web-based user interface will allow users to connect their MetaMask wallets.
- Users can initiate transactions, view transaction history, and manage their accounts.

d. MetaMask Integration:

- MetaMask will be integrated into the web interface, enabling secure transaction signing and wallet management.

e. Chain Connect Nodes:

- Connect to Ethereum nodes, such as Infura or other providers, to access the Ethereum network reliably.

f. Timestamps:

- Transactions will be timestamped using block timestamps or an external oracle service to ensure data integrity.

g. Security Measures:

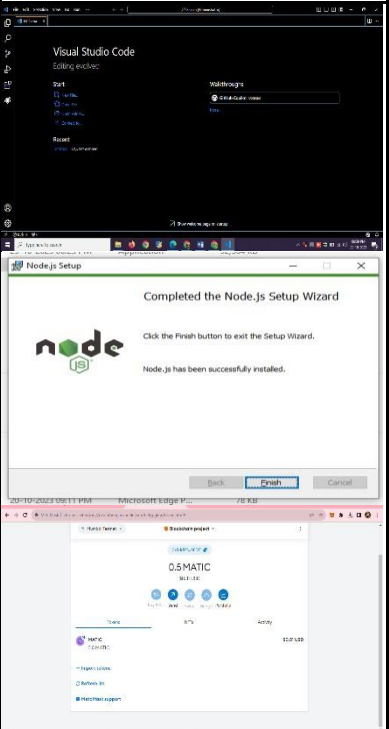
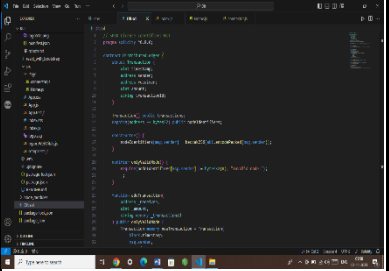
- Robust security measures will be implemented to protect user accounts and sensitive data.

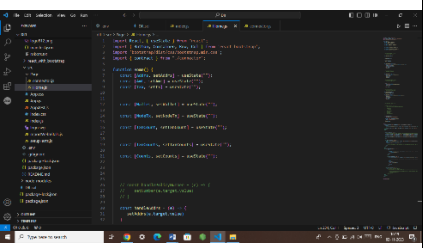
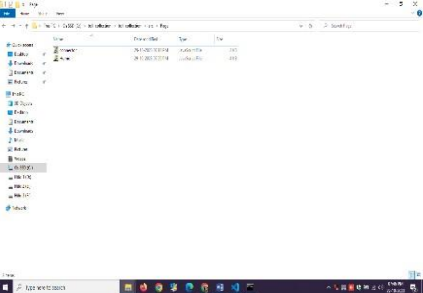
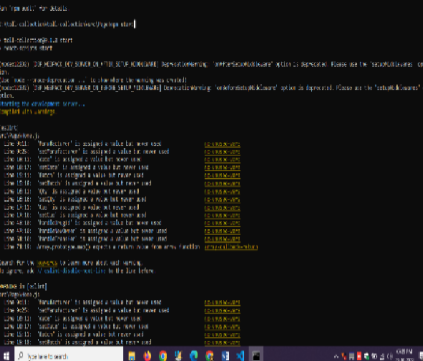
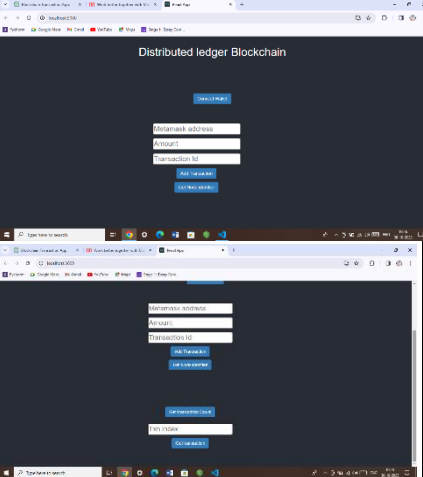
h. Testing and Deployment:

- The application will be thoroughly tested on test networks before being deployed on the Ethereum mainnet.

8. PERFORMANCE TESTING:

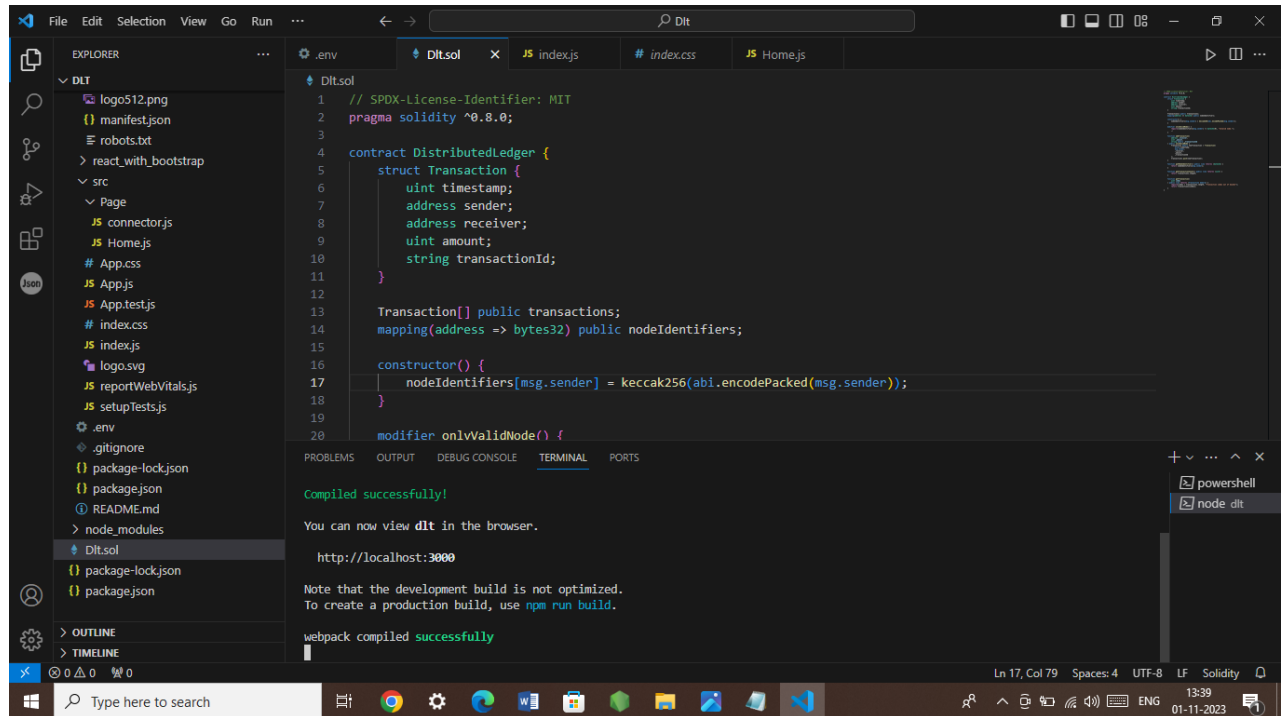
8.1 PERFORMANCE METRICS:

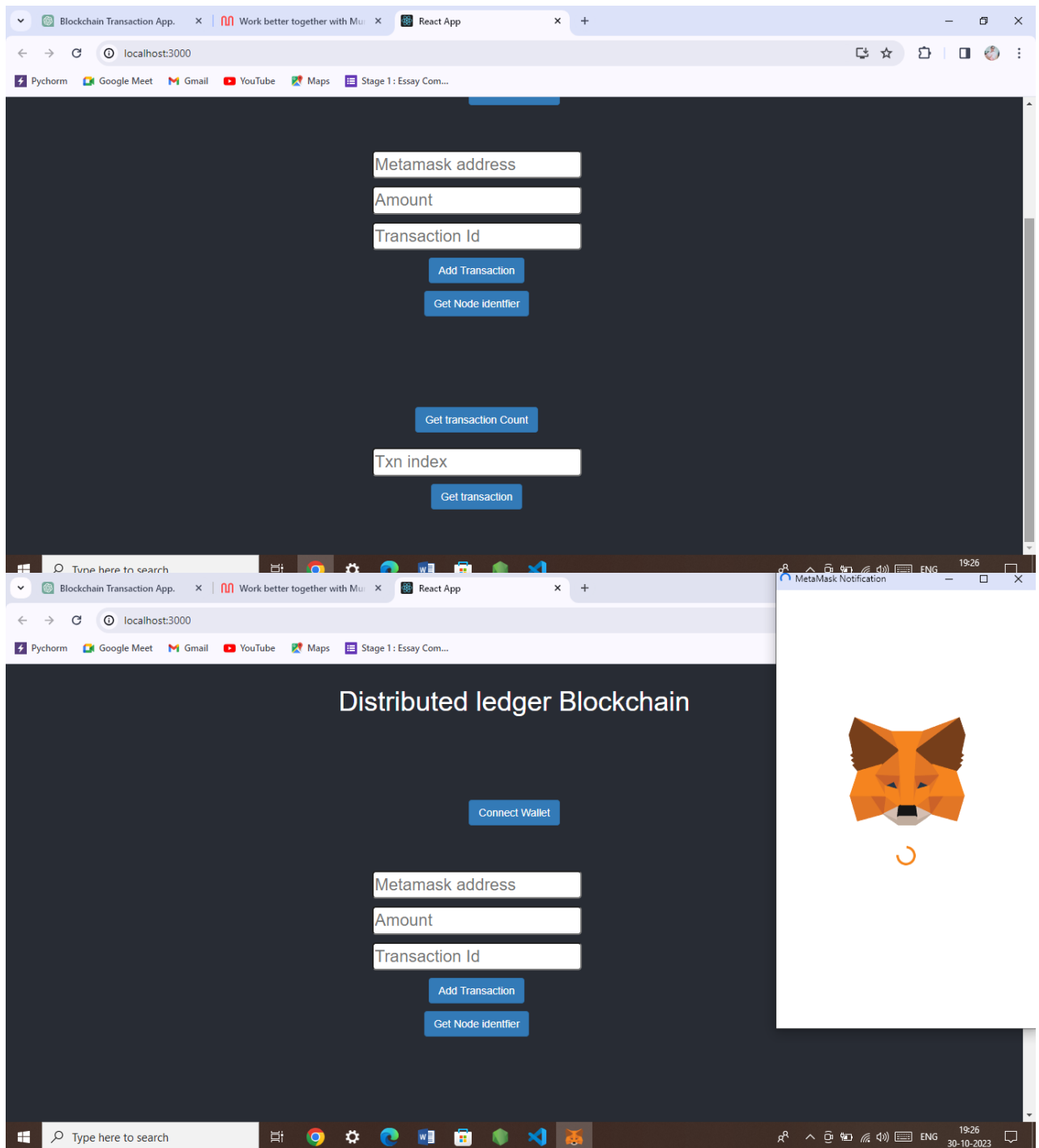
S.No.	Parameter	Values	Screenshot
1.	Information gathering	Setup all the Prerequisite:	
2.	Extract the zip files	Open to VS Code	

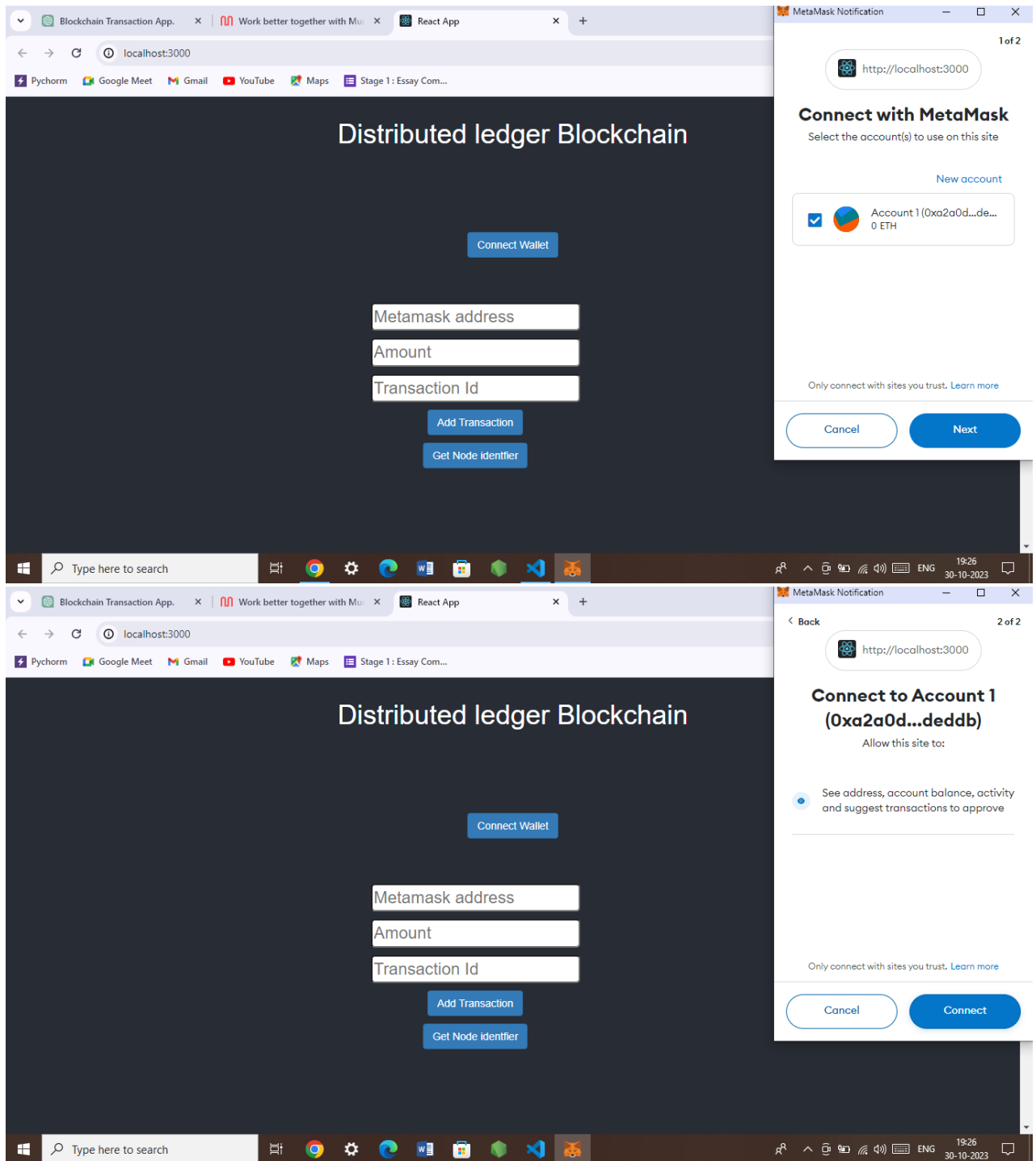
3.	VS Code platform exploring	<ul style="list-style-type: none"> Deploy the Distributed Ledger Blockchain code Debug and run the transaction - inject the MetaMask. 	
4	Open file explorer	<ul style="list-style-type: none"> Open the extracted file and click on the folder. Open src, and search for utilities. Open cmd or powershell or VS Code terminal enter commands: <ol style="list-style-type: none"> npm install npm bootstrap npm start 	 
5	{LOCALHOST ADDRESS	Copy the address and open it to chrome or type npm start in VS Code terminal so you can see the front end of your project.	

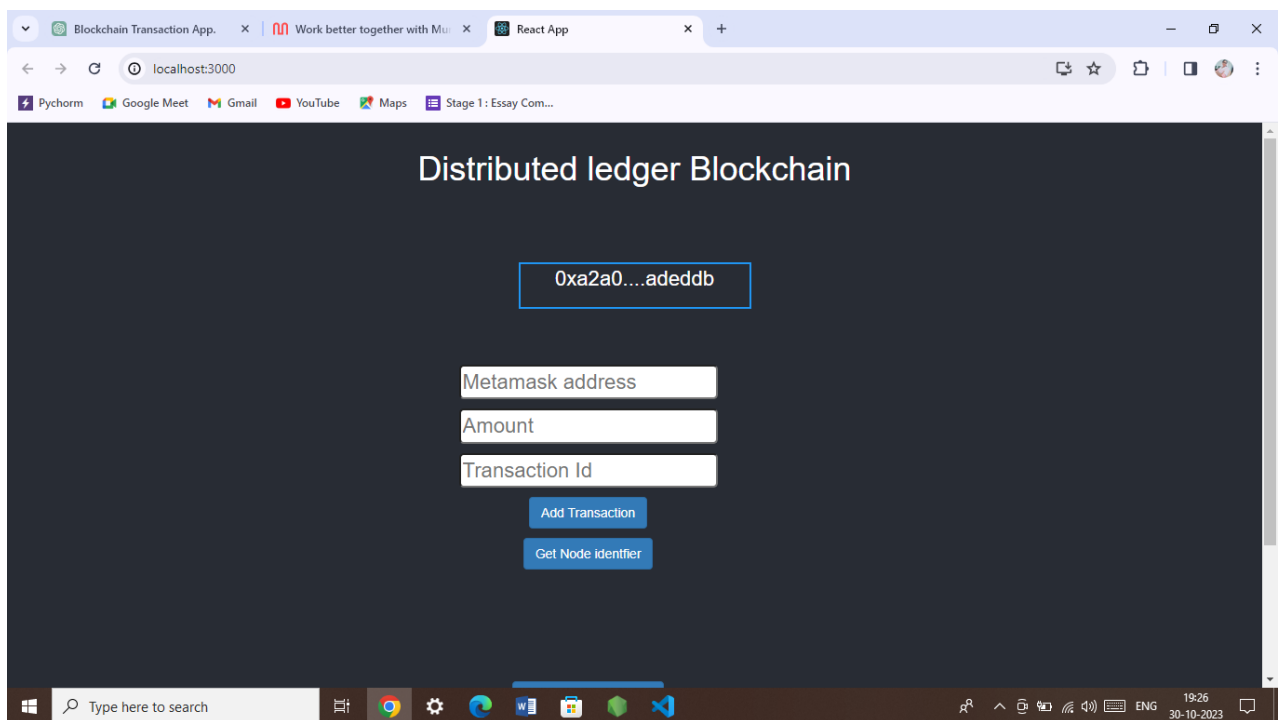
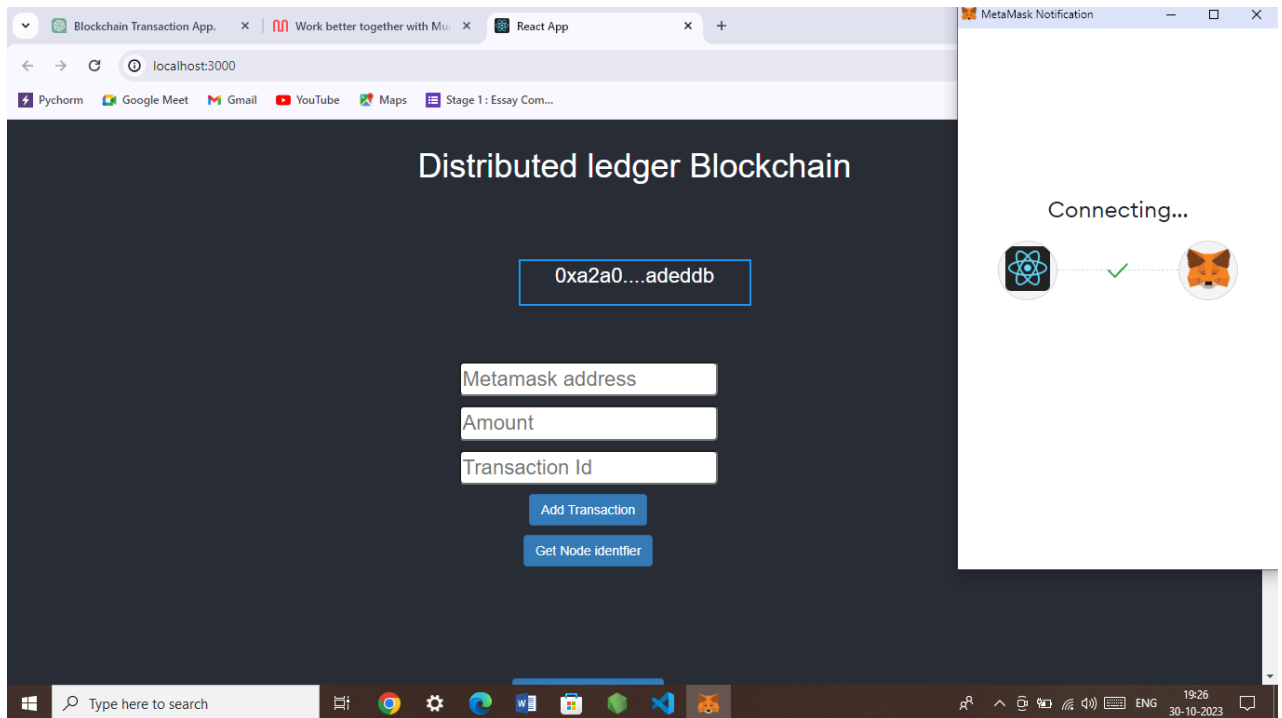
9. RESULT:

9.1 OUTPUT SCREENSHOTS:









10. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- 1. Security and Transparency:** Utilizes blockchain technology to ensure high-level security and transparency, reducing the risk of fraud and unauthorized access.
- 2. Cost Efficiency:** Eliminates the need for traditional intermediaries, reducing transaction fees and increasing cost-effectiveness.
- 3. Timestamping:** Provides immutable timestamps for all transactions, enhancing accountability and record-keeping.
- 4. User-Friendly:** Integrates with popular wallets like MetaMask, ensuring a user-friendly and familiar interface.
- 5. Compliance:** Supports regulatory compliance with features like Know Your Customer (KYC) and Anti-Money Laundering (AML) checks.
- 6. Scalability:** Designed to handle a growing number of users and transactions as the system expands.
- 7. Efficient Documentation:** Comprehensive user guides and documentation enhance user understanding and ease of use.
- 8. Decentralization:** Removes reliance on centralized authorities, offering a trustless environment for financial transactions.

DISADVANTAGES

- 1. Complexity:** Implementing blockchain technology can be complex, requiring specialized skills and expertise, which may lead to development challenges and delays.
- 2. Scalability Concerns:** As the user base grows, the blockchain network may face scalability issues, potentially leading to slower transaction processing times.
- 3. Regulatory Challenges:** Compliance with ever-evolving legal and regulatory requirements, especially in the financial sector, can be a challenge and may require ongoing adjustments.

- 4. User Adoption:** Blockchain technology and cryptocurrency wallets may be unfamiliar to some users, potentially posing a barrier to adoption.
- 5. Security Risks:** While blockchain is generally secure, there are still potential vulnerabilities, such as smart contract bugs and external attacks, which must be addressed.
- 6. Cost of Gas Fees:** Users may incur gas fees for transaction processing on the blockchain, which can vary based on network congestion and market conditions.
- 7. Data Privacy:** Blockchain's transparency may conflict with certain data privacy requirements, necessitating careful handling of sensitive information.
- 8. Maintenance and Upkeep:** Ongoing maintenance and updates to ensure system security and efficiency are necessary and can incur additional costs.
- 9. Dependence on External Tools:** Integration with third-party tools like MetaMask and Chain Connect nodes introduces dependencies that may affect the project's reliability.
- 10. Learning Curve:** Users and administrators may need time to understand and adapt to the blockchain-based system, potentially causing a learning curve.
- 11. Resource Intensive:** Running and maintaining blockchain nodes and infrastructure can be resource-intensive in terms of hardware, energy, and technical expertise.

It's essential to consider these potential disadvantages and address them proactively to ensure the success and sustainability of the project.

11. CONCLUSION:

In conclusion, the Distributed Ledger Blockchain application for financial transactions, designed using Solidity and Node.js, integrated with MetaMask and Chain Connect nodes, offers a promising solution for secure and transparent financial transactions. It leverages the advantages of blockchain technology to enhance security, reduce costs, and provide immutable timestamping. However, it also comes with challenges such as regulatory compliance, scalability issues, and a learning curve for users. Addressing these challenges proactively and ensuring ongoing maintenance and support will be essential for the project's long-term success. With careful planning and execution, this project has the potential to revolutionize the way financial transactions are conducted, promoting trust and efficiency in the digital financial landscape.

11. FUTURE SCOPE:

The future scope of the Distributed Ledger Blockchain application, developed using Solidity and Node.js, and integrated with MetaMask and Chain Connect nodes, is highly promising and multifaceted. It extends to a wide array of possibilities, including the expansion of financial services to encompass lending, savings, and investment products, positioning the project as a comprehensive financial platform. Furthermore, it can explore cross-chain integration, enabling interoperability with multiple blockchain networks and fostering collaboration within the blockchain ecosystem. Embracing the burgeoning decentralized finance (DeFi) sector is another avenue, allowing for integration with DeFi protocols and the introduction of decentralized financial products.

Tokenization of real-world assets, such as real estate or art, offers the potential for fractional ownership and investment diversification. Cross-border payments and blockchain-based identity management are areas for growth, while governance through decentralized autonomous organizations (DAOs) can empower the community. Advancements in security, regulatory compliance services, artificial intelligence-driven analytics, and mobile applications are essential for staying competitive. Collaboration with industry stakeholders, sustainability initiatives, research and development, and educational outreach further shape the project's future. In sum, this project is poised to adapt and innovate, making a substantial impact on the financial and blockchain landscape.

12. APPENDIX:

Source Code: Dlt.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DistributedLedger {
    struct Transaction {
        uint timestamp;
        address sender;
        address receiver;
        uint amount;
        string transactionId;
    }
}
```

```

Transaction[] public transactions;
mapping(address => bytes32) public nodeIdentifiers;
constructor() {
    nodeIdentifiers[msg.sender] = keccak256(abi.encodePacked(msg.sender));
}

modifier onlyValidNode() {
    require(nodeIdentifiers[msg.sender] != bytes32(0), "Invalid node.");
    _;
}

function addTransaction(
    address _receiver,
    uint _amount,
    string memory _transactionId
) public onlyValidNode {
    Transaction memory newTransaction = Transaction(
        block.timestamp,
        msg.sender,
        _receiver,
        _amount,
        _transactionId
    );
    transactions.push(newTransaction);
}

function getNodeIdentifier() public view returns (bytes32) {
    return nodeIdentifiers[msg.sender];
}

function getTransactionCount() public view returns (uint) {
    return transactions.length;
}

```



```

    }

    function getTransaction(
        uint index
    ) public view returns (Transaction memory) {
        require(index < transactions.length, "Transaction index out of bounds");
        return transactions[index];
    }
}

```

Home.js and Connector.js:

```

//Home.js
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "../connector";

function Home() {
    const [Addr, setAddr] = useState("");
    const [Amt, setAmt] = useState("");
    const [txx, settx] = useState("");

    const [Wallet, setWallet] = useState("");

    const [NodeTx, setNodeTx] = useState("");

    const [TxnCount, setTxnCount] = useState("");

    const [TxnCounts, setTxnCounts] = useState("");

    const [Counts, setCounts] = useState("");

    // const handlePolicyNumber = (e) => {
    //     setNumber(e.target.value)
    // }

    const handleAddr = (e) => {
        setAddr(e.target.value)
    }
    const handleAmt = (e) => {
        setAmt(e.target.value)
    }
    const handleTxn = (e) => {

```

```

        settx(e.target.value)
    }
    const handleRegAsset = async () => {
        try {
            let tx = await contract.addTransaction(Addrs, Amt.toString(), txx)
            let wait = await tx.wait()
            alert(wait.transactionHash)
            console.log(wait);
        } catch (error) {
            alert(error)
        }
    }
    // const handlePublishHash = (e) => {
    //     setPubHash(e.target.value)
    // }

    const handlePublish = async () => {
        try {
            let tx = await contract.getNodeIdentifier()
            console.log(tx);
            setNodeTx(tx)
        } catch (error) {
            alert(error)
        }
    }

    // const handleUnPublishHash = (e) => {
    //     setUnPubHash(e.target.value)
    // }

    const handleUnPublish = async () => {
        try {
            let tx = await contract.getTransactionCount()
            setTxnCount(tx)
            console.log(tx);

        } catch (error) {
            alert(error)
        }
    }

    // const handleTransferHash = (e) => {
    //     setTransferHash(e.target.value)
    // }

    const handleTxnCount = (e) => {

```

```

    setTxnCounts(e.target.value)

const handleTxncnt = async () => {
  try {
    let tx = await contract.getTransaction(TxnCounts.toString())
    setCounts(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

// const handleGetIds = async (e) => {
//   setGIds(e.target.value)
// }
// const handleGetDetails = async () => {
//   try {
//     let tx = await contract.digitalAssets(gId.toString())
//     let arr = []
//     tx.map(e => {
//       arr.push(e)
//     })
//     console.log(tx);
//     setDetails(arr)
//   } catch (error) {
//     alert(error)
//     console.log(error);
//   }
// }

const handleWallet = async () => {
  if (!window.ethereum) {
    return alert('please install metamask');
  }
  const addr = await window.ethereum.request({
    method: 'eth_requestAccounts',
  });

  setWallet(addr[0])
}

return (
  <div>
    <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Distributed ledger
    Blockchain</h1>
    {!Wallet ?

```

```

        <Button onClick={handleWallet} style={{ marginTop: "30px",
marginBottom: "50px" }}>Connect Wallet </Button>
        :
        <p style={{ width: "250px", height: "50px", margin: "auto",
marginBottom: "50px", border: '2px solid #2096f3' }}>{Wallet.slice(0,
6)}...{Wallet.slice(-6)}</p>
        }
    <Container>
        <Row>
            <Col style={{marginRight:"100px"}}>
                <div>
                    {/* <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePolicyNumber} type="string" placeholder="Policy number"
value={number} /> <br /> */}
                    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAddrs} type="string" placeholder="Metamask address"
value={Addrs} /> <br />

                    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAmt} type="number" placeholder="Amount" value={Amt} /> <br />

                    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTxn} type="string" placeholder="Transaction Id" value={txx}
/> <br />
                    <Button onClick={handleRegAsset} style={{ marginTop: "10px" }}
variant="primary">Add Transaction</Button>

                </div>
            </Col>

            <Col style={{ marginRight: "100px" }}>
                <div>
                    <Button onClick={handlePublish} style={{ marginTop: "10px" }}
variant="primary"> Get Node identifier</Button>
                    {NodeTx ?
                        <p>{NodeTx.slice(0,6)}...{NodeTx.slice(-4)}</p>
                        : <p></p>
                    }

                    {/* <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePublishHash} type="string" placeholder="Asset Hash"
value={PubHash} /> <br /> */}

                </div>
            </Col>
        </Row>
        <Row style={{marginTop:"100px"}}>
            <Col style={{ marginRight: "100px" }}>
                <div>

```

```

        <Button onClick={handleUnPublish} style={{ marginTop:
"10px" }} variant="primary"> Get transaction
Count</Button>

        {TxnCount?
          <p>{TxnCount.toString()}</p>
          : <p></p>
        }
      </div>
    </Col>
    <Col style={{ marginRight: "100px" }}>
      <div>
        <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTxnCount} type="number" placeholder="Txn index"
value={TxnCounts} /> <br />

        <Button onClick={handleTxncnt} style={{ marginTop: "10px"
}} variant="primary"> Get transaction</Button>
        {Counts ?
          Counts?.map(e => {
            return <p>{e.toString()}</p>
          })
          : <p></p>
        }
      </div>
    </Col>
  </Row>
  <Row style={{ marginTop: "50px" }}>
    </* <Col style={{ marginRight: "100px" }}>
      <div style={{ margin: "auto", marginTop: "100px" }}>
        <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleGetIds} type="string" placeholder="Enter Assets Hash"
value={gId} /><br />

        <Button onClick={handleGetDetails} style={{ marginTop:
"10px" }} variant="primary">Get Digital Assets</Button>
        {Details ? Details?.map(e => {
          return <p>{e.toString()}</p>
        }) : <p></p>
        }
      </div>
    </Col>    */>

  </Row>
</Container>

</div>
)
}

export default Home;

```

```
//Conector.js
const { ethers } = require("ethers");
const abi = [
  {
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "_receiver",
        "type": "address"
      },
      {
        "internalType": "uint256",
        "name": "_amount",
        "type": "uint256"
      },
      {
        "internalType": "string",
        "name": "_transactionId",
        "type": "string"
      }
    ],
    "name": "addTransaction",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "getNodeIdentifier",
    "outputs": [
      {
        "internalType": "bytes32",
        "name": "",
        "type": "bytes32"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
```

```

        "name": "index",
        "type": "uint256"
    }
],
"name": "getTransaction",
"outputs": [
    {
        "components": [
            {
                "internalType": "uint256",
                "name": "timestamp",
                "type": "uint256"
            },
            {
                "internalType": "address",
                "name": "sender",
                "type": "address"
            },
            {
                "internalType": "address",
                "name": "receiver",
                "type": "address"
            },
            {
                "internalType": "uint256",
                "name": "amount",
                "type": "uint256"
            },
            {
                "internalType": "string",
                "name": "transactionId",
                "type": "string"
            }
        ],
        "internalType": "struct DistributedLedger.Transaction",
        "name": "",
        "type": "tuple"
    }
],
"stateMutability": "view",
"type": "function"
},
{
    "inputs": [],
    "name": "getTransactionCount",
    "outputs": [
        {
            "internalType": "uint256",

```

```
    "name": "",
    "type": "uint256"
  }
],
"stateMutability": "view",
"type": "function"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "name": "nodeIdentifiers",
  "outputs": [
    {
      "internalType": "bytes32",
      "name": "",
      "type": "bytes32"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "transactions",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "timestamp",
      "type": "uint256"
    },
    {
      "internalType": "address",
      "name": "sender",
      "type": "address"
    }
  ],
  {
    "internalType": "address",
```



```

        "name": "receiver",
        "type": "address"
    },
    {
        "internalType": "uint256",
        "name": "amount",
        "type": "uint256"
    },
    {
        "internalType": "string",
        "name": "transactionId",
        "type": "string"
    }
],
"stateMutability": "view",
"type": "function"
}
]

if (!window.ethereum) {
    alert('Meta Mask Not Found')
    window.open("https://metamask.io/download/")
}

export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0x0A7FCbE739c0E9Ac3c4d4e85b67Bb3e31e31884e"

export const contract = new ethers.Contract(address, abi, signer)

```

Github Link :

<https://github.com/Shruthi011002/NM2023TMID01745>

Project Demo Link:

Demo video link:

https://drive.google.com/file/d/1Jl6rPHyr9UvKLZ4kA8Q_8cy77JDRuSDP/view?usp=drivesdk

