

PROJECT REPORT

CENTRAL BANK SMART CONTRACT USING BLOCKCHAIN

1.INTRODUCTION:

1.1 PROJECT OVERVIEW:

The Central Bank Smart Contract Using Blockchain project is a groundbreaking initiative designed to modernize and streamline central banking operations. By leveraging blockchain technology and smart contracts, this project seeks to enhance the transparency, security, and efficiency of central bank functions. It aims to automate critical aspects of monetary policy, such as interest rate management and currency issuance, while ensuring the immutability of data and improving auditability. This initiative also explores the potential introduction of a central bank digital currency (CBDC) to facilitate secure and efficient digital transactions. By implementing regulatory smart contracts and data analytics, the central bank can enforce compliance with financial regulations and gain real-time insights into economic indicators. The project aspires to achieve heightened security, cost savings, and real-time monitoring for sound policy decision-making while addressing challenges related to regulation, privacy, scalability, and public adoption.

1.2 PURPOSE:

The purpose of implementing a Central Bank Smart Contract Using Blockchain is multifaceted and includes:

Enhancing Transparency: Blockchain technology increases the transparency and auditability of central bank operations, as all transactions are recorded on an immutable ledger accessible to relevant parties.

Efficient Monetary Policy Execution: Smart contracts enable automated execution of monetary policies, including interest rate adjustments and currency issuance, streamlining the decision-making process and implementation.

Improving Security: Blockchain's cryptographic and decentralized nature enhances the security of financial transactions, reducing the risk of fraud, cyberattacks, and data breaches.

2. LITERATURE SURVEY:

Existing problem:

One significant challenge in implementing a Central Bank Smart Contract Using Blockchain is the issue of regulatory compliance and the development of a legal framework that accommodates this innovative technology. Blockchain operates on a decentralized and often pseudonymous ledger, which can make it challenging to enforce existing financial regulations and monitor transactions effectively.

The nature of blockchain's immutability can also pose challenges in terms of rectifying errors or addressing illicit activities. Moreover, the legal status and recognition of smart contracts and digital currencies in various jurisdictions can vary, leading to regulatory uncertainties. Central banks need to work closely with government authorities, lawmakers, and international bodies to establish a comprehensive legal and regulatory framework that aligns with the technology's potential and ensures a secure, transparent, and compliant financial ecosystem. This regulatory hurdle underscores the need for careful consideration and collaboration in the development and deployment of blockchain-based solutions in the central banking sector.

References:

1. Adrian, T., & Mancini-Griffoli, T. (2018). Central Bank Digital Currency: Design Principles and Balance Sheet Implications. International Monetary Fund (IMF).
2. Bech, M. L., & Garratt, R. (2017). Central Bank Cryptocurrencies. Bank for International Settlements (BIS).
3. Deloitte. (2017). Smart Contracts: Challenges and Opportunities for Financial Markets. Deloitte.
4. World Economic Forum. (2016). Distributed Ledger Technology in Payments, Clearing, and Settlement. World Economic Forum.
5. Mills, D., & Wang, K. W. (2016). Blockchain and Financial Market Innovation. Federal Reserve Board.

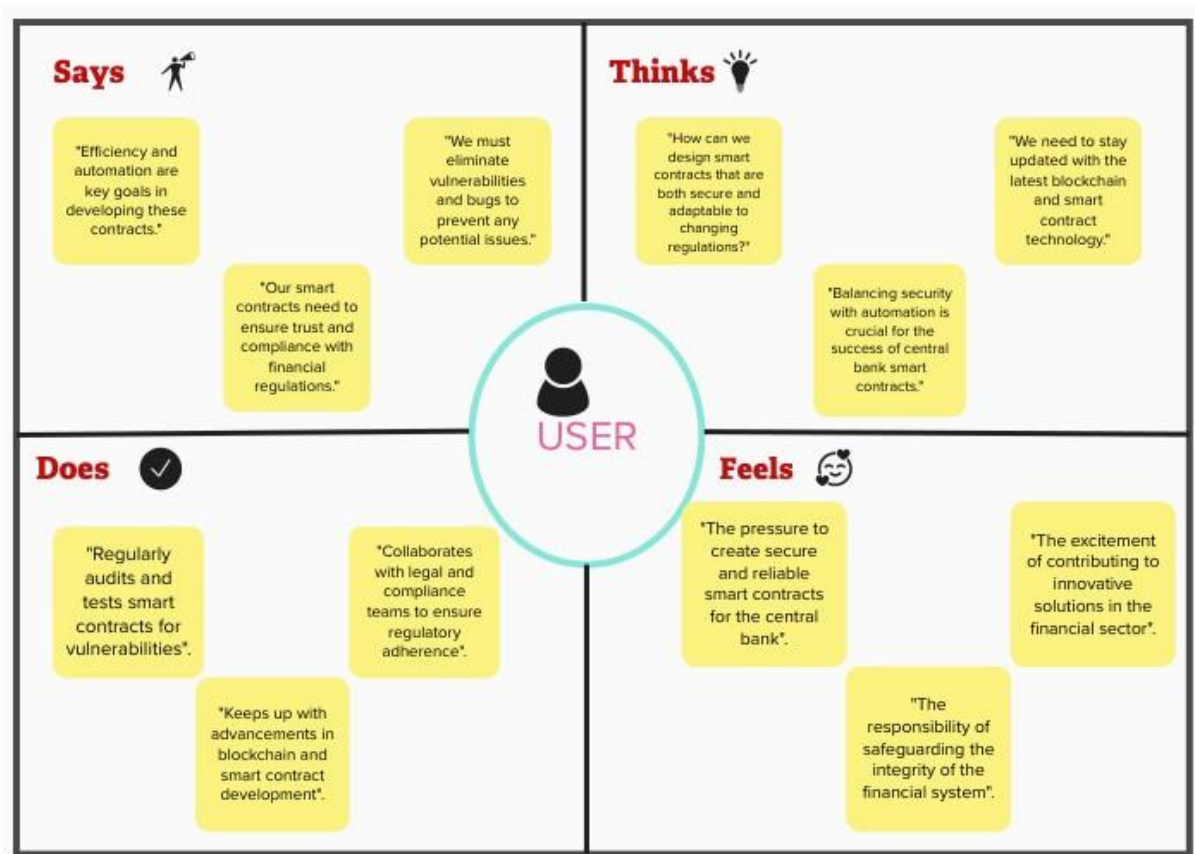
2.2 Problem Statement Definition:

The Central Bank Smart Contract is designed to leverage blockchain technology for building a decentralized banking system. This smart contract allows users to perform banking transactions securely, transparently, and efficiently through the power of blockchain. Blockchain technology came into the ground to overcome these issues. It offers decentralized nodes for the banking system and is used to produce a transparent banking system for its end-to-end verification advantages. This technology is a replacement for the traditional banking system with distributed, non-repudiation, and security protection characteristics. You are a Blockchain expert in a major corporate bank and have been tasked to create a smart contract to perform banking transactions. Create a Smart Contract for a banking.

The problem statement for the Central Bank Smart Contract using blockchain is a concise declaration of the existing challenges and deficiencies within the traditional banking system that necessitate the integration of blockchain technology. It underscores issues such as limited transparency, high operational costs, sluggish transaction processing, security vulnerabilities, and the imperative for a more efficient, secure, and decentralized financial infrastructure. This problem statement forms the basis for the development of the smart contract, providing a clear direction by delineating the specific problems to be addressed and the overarching objectives in leveraging blockchain to establish a more robust and transparent central banking system.

3.IDEATION &PROPOSED SOLUTION:

Empathy Map Canvas:



Ideation & Brainstorming:



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](#) →



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

The Central Bank Smart Contract is designed to leverage blockchain technology for building a decentralized banking system. This smart contract allows users to perform banking transactions securely, transparently, and efficiently through the power of blockchain. Blockchain technology came into the ground to overcome these issues. It offers decentralized nodes for the banking system and is used to produce a transparent banking system for its end-to-end verification advantages.



Key rules of brainstorming

To run a smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.



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 switch to switch/lock to start drawing.

Person 1

Multi-Signature Wallet Smart Contract

Tokenized Assets and Asset Management

Delegated Staking and Governance Contract

Person 2

KYC and AML Verification Smart Contract

Privacy-Preserving Transactions

Audit and Compliance Monitoring

Person 3

Mobile Banking Application Integration

Educational Resources and Support

User-Friendly Transaction History

Person 4

Project Roadmap and Milestone Planning

Budget and Resource Management

Testing and Deployment Strategy



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 customizable tags to sticky notes to make it easy to find, browse, organize, and categorize important ideas or themes within your mural.

Multi-Signature Wallet Smart Contract:

Develop a smart contract for multi-signature wallets to enable shared account management securely.

Enhance security by requiring multiple signatures for transaction approval. Tokenized Assets and Asset Management:

Create a smart contract for tokenizing traditional assets (e.g., real estate, stocks) on the blockchain.

Facilitate secure buying and selling of tokenized assets within the decentralized banking system.

4

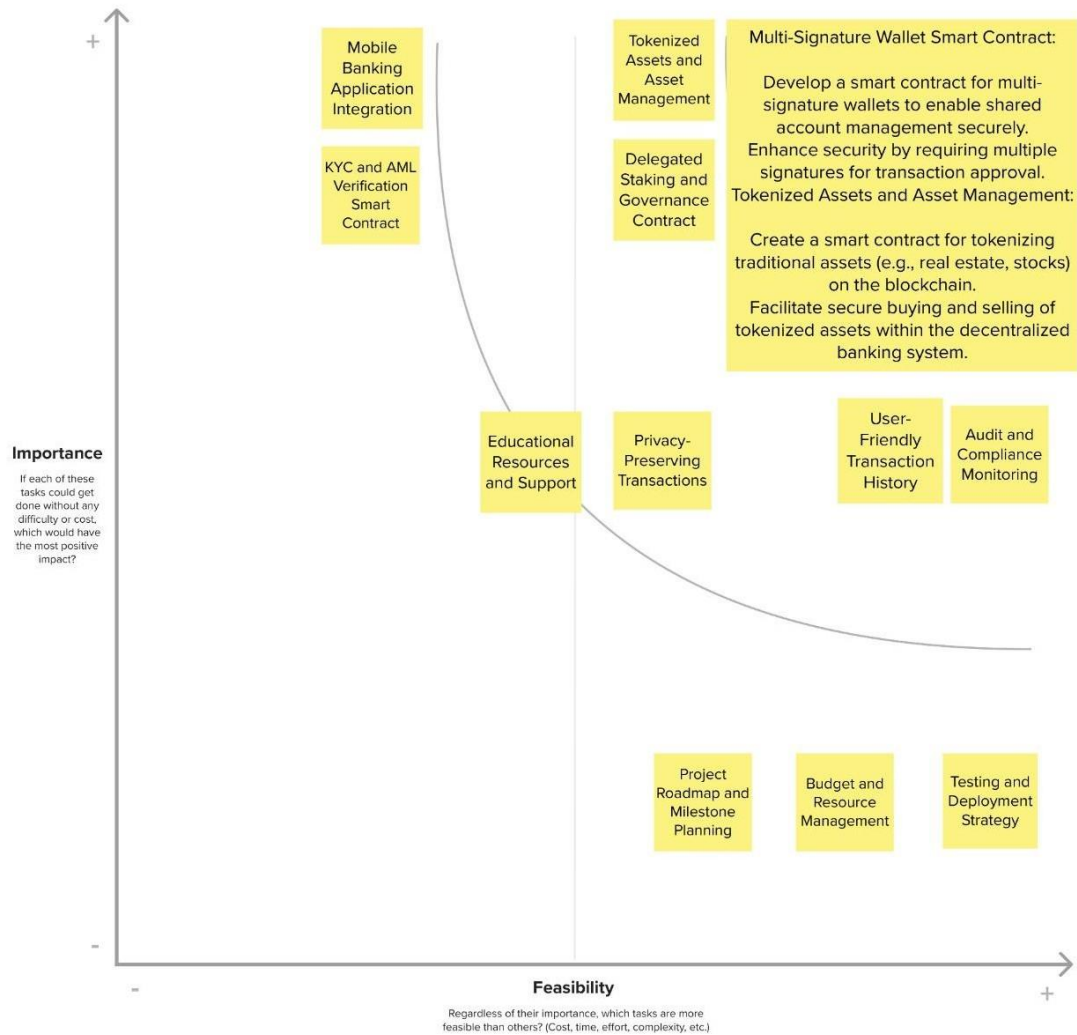
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

🕒 20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H** key on the keyboard.



4.REQUIREMENTANALYSIS:

Functional requirement:

User Registration and Authentication:

- Users should be able to register securely on the platform using blockchain-based identity verification.
- Implement multi-factor authentication for enhanced security.

Account Management:

- Users should be able to create and manage different types of accounts (e.g., savings, checking, investment) on the blockchain.
- Allow for account updates and maintenance.

Transaction Processing:

- Enable various transaction types, including deposits, withdrawals, transfers, and loan requests.
- Implement real-time transaction confirmation and settlement.

Smart loan and Credit scoring:

- Allow users to apply for loans, with the smart contract assessing creditworthiness based on blockchain data.
- Automatically disburse loans and handle repayments.

Token Management:

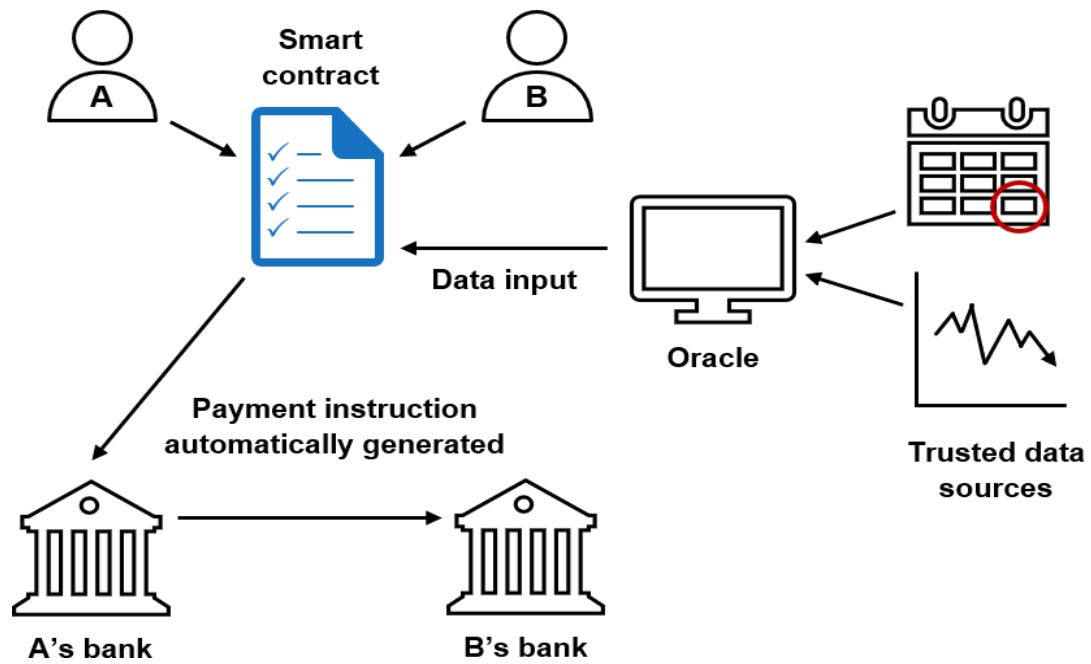
- Automatically calculate and credit interest on savings and investment accounts.
- Implement token minting, transfer, and burning functionalities.

Non-Functional requirements:

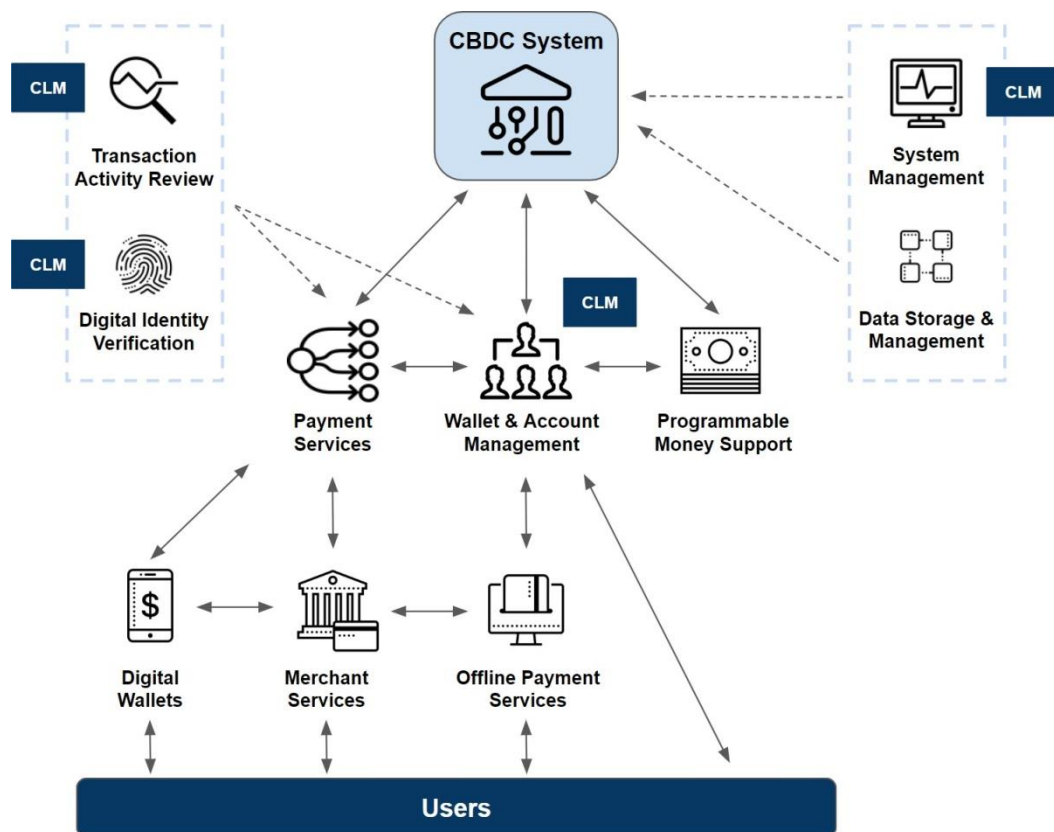
- Non-functional requirements for a Central Bank Smart Contract using blockchain are essential considerations that focus on how the system operates rather than specific functionalities.
- The smart contract should exhibit high availability, ensuring it is accessible to users around the clock, with minimal downtime for maintenance or upgrades. Scalability is crucial to accommodate a growing user base and increasing transaction volumes.
- Robust security measures must be in place to protect sensitive user data and assets. Additionally, the system should offer high performance, with quick transaction processing and minimal latency. Compliance with regulatory standards and industry best practices is non-negotiable, ensuring the platform adheres to legal and ethical standards.
- Moreover, user-friendly and intuitive user interfaces, as well as comprehensive documentation and support, are vital to enhance the user experience. Efficient resource utilization and cost-effectiveness must also be considered to ensure the system remains economically viable. Overall, these non-functional requirements are pivotal in shaping the success, reliability, and sustainability of a Central Bank Smart Contract using blockchain.

5. PROJECT DESIGN:

Data Flow Diagram:

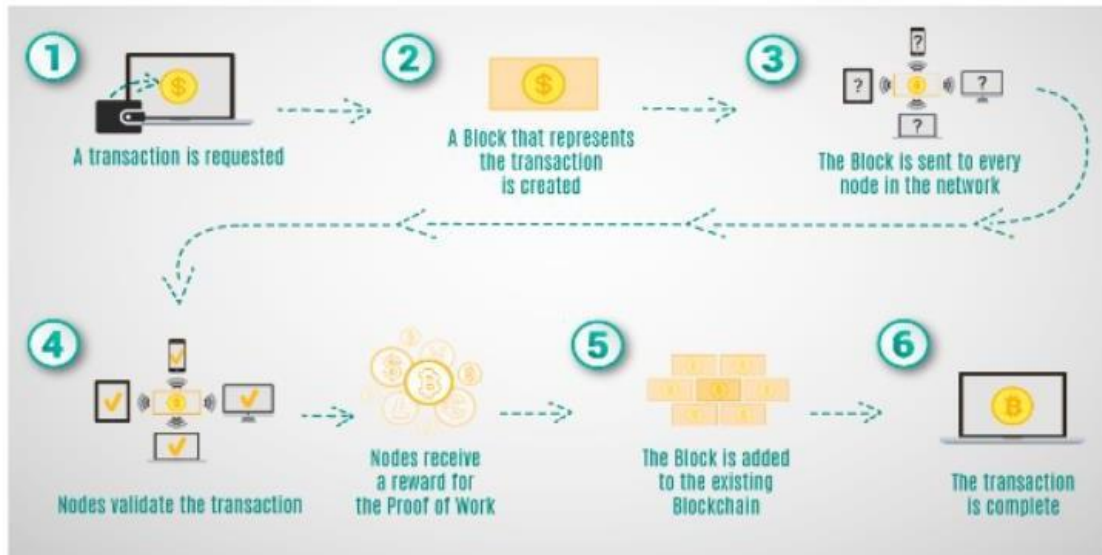


Solution Architecture:



6 PROJECT PLANNING:

Technical Architecture:



7.CODING&SOLUTIONING:

Preliminary Definitions:

1. **SPDX-License-Identifier:** This is a standardized identifier for specifying the license under which the contract's code is released. In this case, it uses the MIT License, which is an open-source license.

// SPDX-License-Identifier: MIT

```
pragma solidity ^0.8.0;
```

2. **Pragma Directive:** This specifies the version of the Solidity compiler to be used for compiling the code. Here, it's Solidity version 0.8.0.

3. **Contract declaration:** This declares a Solidity smart contract named "Bank." It will encapsulate the logic and state of a simple banking system.

```
contract Bank {  
    address public owner;  
    mapping(address => uint256) public balances;
```

4. **State variables:** address public owner: This state variable stores the Ethereum address of the contract owner, and it's publicly accessible.

5. `mapping(address => uint256) public balances;` This state variable is a mapping that associates Ethereum addresses (users) with their account balances. It's publicly accessible, allowing anyone to query balances.

```
constructor() {  
    owner = msg.sender;  
}
```

6. Constructor: The constructor is a special function that gets executed only once when the contract is deployed. In this case, it sets the owner variable to the Ethereum address of the person or entity who deploys the contract, which is stored in `msg.sender`. This establishes the contract owner.

7. Modifier: The `onlyOwner` modifier is defined to restrict access to certain functions. It checks if the sender of the transaction is the contract owner (as defined in the constructor). If not, it raises an exception with the error message provided. If the check passes, it continues with the function execution (indicated by `_;`).

```
modifier onlyOwner() {  
    require(msg.sender == owner, "Only contract owner can call this");  
    _;  
  
}
```

8. Function - `mintMoney`: This function allows the contract owner to add funds to their own balance. It checks that the amount is greater than zero and then increments the balance of the owner.

```
function mintMoney(uint256 amount) external onlyOwner {  
    require(amount > 0, "Amount must be greater than 0");  
    balances[msg.sender] += amount;  
  
}
```

9. Function - `withdrawMoney`: This function allows any address to withdraw funds from their balance. It checks that the account balance is sufficient to cover the withdrawal, and if so, it deducts the specified amount from the sender's balance.

```
function withdrawMoney(uint256 amount) external {  
    require(balances[msg.sender] >= amount, "Insufficient balance");  
    balances[msg.sender] -= amount;  
  
}
```

10.Function - transferFunds: This function allows the contract owner to transfer funds from their account to another account (specified by recipientAddress). It checks the owner's balance and then transfers the specified _amount from the owner's account to the recipient's account.

```
function transferFunds(address payable recipientAddress, uint _amount) public  
onlyOwner {  
    require(balances[msg.sender] >= _amount, "Insufficient balance");  
    balances[msg.sender] -= _amount;  
  
    balances[recipientAddress] += _amount;  
  
}
```

11.Function - checkBalance: This function allows any address to query their account balance. It is a read-only function marked with the view keyword, indicating that it does not modify the contract state. It returns the balance associated with the caller's address.

```
function checkBalance() external view returns (uint256) {  
    return balances[msg.sender];  
}
```

12. Key Concepts:

Events: This contract does not include any events. Events are typically used to log significant contract actions for external consumption and analysis.

Modifiers: The onlyOwner modifier is used to control access to functions based on the caller's status as the contract owner.

Constructor: The constructor is used to initialize the contract's state variables and runs only once during contract deployment.

Functions: The contract contains functions for minting money, withdrawing money, transferring funds, and checking balances. These functions define the core operations of the banking system.

State Variables: The owner and balances variables are used to store the state of the contract. owner represents the contract owner, and balances maps Ethereum addresses to their account balances.

7.1 Feature:

The central bank smart contract leverages blockchain technology to create a decentralized banking system, eliminating the need for intermediaries and enhancing transparency. Users have more control over their financial assets and can access banking services securely and conveniently, promoting financial inclusion and accessibility.

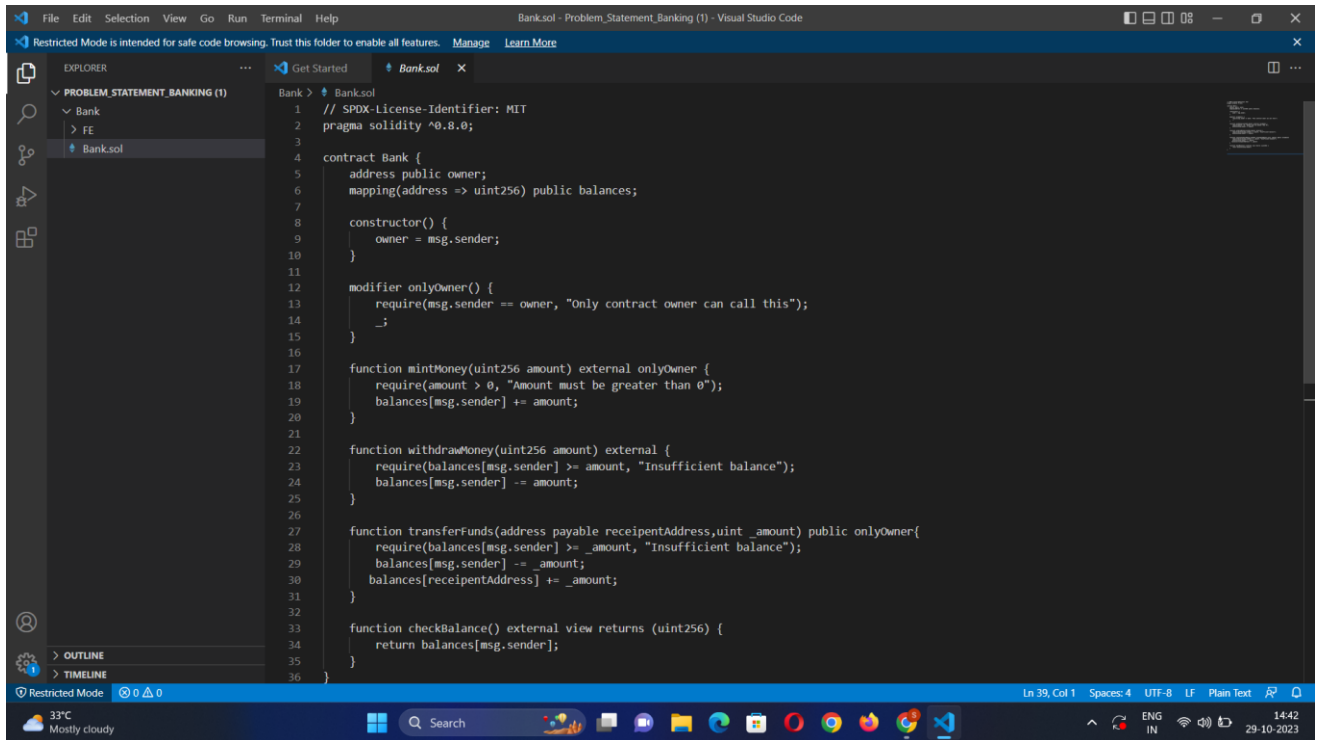
8.PERFORMANCE TESTING

| S.No. | Parameter | Values | Screenshot |
|-------|------------------------------|--|---|
| 1. | Information gathering | Set up all the Prerequisite: |  |
| 2. | Extract the zip files | Open to vscode |  |
| 3. | Remix Ide platform exploring | Deploy the smart contract code Deploy and run the transaction.By selecting the environment – inject the MetaMask. |  |

| | | | |
|---|----------------------------|--|--|
| 4 | Open file explorer | <p>Open the extracted file and click on the folder.</p> <p>Open src, and search for utilities.</p> <p>Open cmd enter commands</p> <ol style="list-style-type: none"> 1. npm install 2. Npm install bootstrap 3. npm start | |
| 5 | {LOCAL HOST IP ADDRESS} | Copy the address and open it to chrome so you can see the frontend of your project. | |

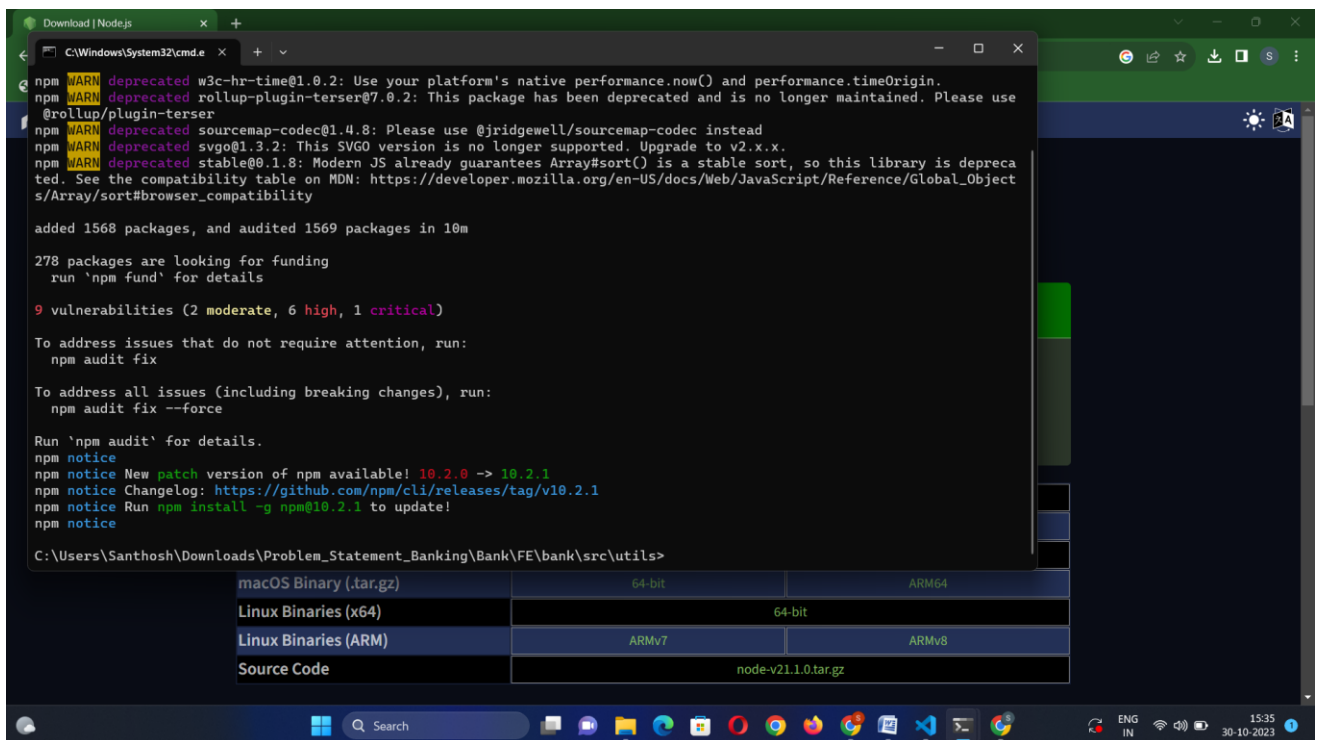
9.RESULTS:

Output Screenshots:



The screenshot shows the Visual Studio Code editor with a file named 'Bank.sol' open. The code is a Solidity smart contract for a bank. It includes a constructor, a modifier for owner-only actions, and functions for minting, withdrawing, and transferring funds, along with a balance-checking function.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Bank {
5     address public owner;
6     mapping(address => uint256) public balances;
7
8     constructor() {
9         owner = msg.sender;
10    }
11
12    modifier onlyOwner() {
13        require(msg.sender == owner, "Only contract owner can call this");
14        _;
15    }
16
17    function mintMoney(uint256 amount) external onlyOwner {
18        require(amount > 0, "Amount must be greater than 0");
19        balances[msg.sender] += amount;
20    }
21
22    function withdrawMoney(uint256 amount) external {
23        require(balances[msg.sender] >= amount, "Insufficient balance");
24        balances[msg.sender] -= amount;
25    }
26
27    function transferFunds(address payable receiptAddress, uint _amount) public onlyOwner {
28        require(balances[msg.sender] >= _amount, "Insufficient balance");
29        balances[msg.sender] -= _amount;
30        balances[receiptAddress] += _amount;
31    }
32
33    function checkBalance() external view returns (uint256) {
34        return balances[msg.sender];
35    }
36 }
```



The screenshot shows a terminal window with the output of an 'npm audit' command. It lists several deprecated packages and their replacements, along with a summary of vulnerabilities found in the project's dependencies.

```
npm WARN deprecated w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
npm WARN deprecated rollup-plugin-terser@7.0.2: This package has been deprecated and is no longer maintained. Please use @rollup/plugin-terser
npm WARN deprecated source-map-codec@1.4.8: Please use @jridgewell/source-map-codec instead
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compatibility

added 1568 packages, and audited 1569 packages in 10m

278 packages are looking for funding
  run 'npm fund' for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
npm notice
npm notice New patch version of npm available! 10.2.0 -> 10.2.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.1
npm notice Run npm install -g npm@10.2.1 to update!
npm notice
```

The terminal also shows the path to the project files: `C:\Users\Santhosh\Downloads\Problem_Statement_Banking\Bank\FE\bank\src\utils>`

| Platform | Architecture | Package |
|----------------------|---------------------|---------|
| macOS Binary | 64-bit | ARM64 |
| Linux Binaries (x64) | 64-bit | ARM64 |
| Linux Binaries (ARM) | ARMv7 | ARMv8 |
| Source Code | node-v21.1.0.tar.gz | |

Naan Mudhalvan Massive Upskill - Student - Remix - Ethereum IDE - MetaMask - WhatsApp

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js

DEPLOY & RUN TRANSACTIONS ✓

ENVIRONMENT

Remix VM (Shanghai)

ACCOUNT

0x5B3...eddC4 (99.9999999%)

GAS LIMIT

3000000

VALUE

0 Wei

CONTRACT

Bank - bank.sol

Deploy

Publish to IPFS

At Address Load contract from Address

Transactions recorded 1

Deployed Contracts

BANK AT 0xD91...39138 (MEMORY)

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Bank {
5     address public owner;
6     mapping(address => uint256) public balances;
7
8     constructor() {
9         owner = msg.sender;
10    }
11
12    modifier onlyOwner() {
13        require(msg.sender == owner, "Only contract owner can call this");
14        _;
15    }
16
17    function mintMoney(uint256 amount) external onlyOwner {
18        require(amount > 0, "Amount must be greater than 0");
19        balances[msg.sender] += amount;
20    }
21
22    function withdrawMoney(uint256 amount) external {
23        require(balances[msg.sender] >= amount, "Insufficient balance");
24        balances[msg.sender] -= amount;
25    }
26
27    function transferFunds(address payable receiptAddress, uint _amount) public onlyOwner {
28        require(balances[msg.sender] >= _amount, "Insufficient balance");
29        balances[msg.sender] -= _amount;
30        balances[receiptAddress] += _amount;
31    }
32
33    function checkBalance() external view returns (uint256) {
34        return balances[msg.sender];
35    }
36}
```

34°C Rain showers

Search

ENG IN 16:52 29-10-2023

empathy map for central - Naan Mudhalvan Massive - MetaMask - Chrome Web - Problem_Statement_Bank - Remix - Ethereum IDE - MetaMask

MetaMask | chrome-extension://nkbihfegoeaaoehfknkodbefggknj/home.html#

METAMASK

Ethereum Mainnet

Account 1

0x4B782...F0768

0 ETH

\$0.00 USD

Buy & S... Send Swap Bridge Portfolio

Tokens

Ethereum 0 ETH

+ Import tokens

Refresh list

MetaMask support

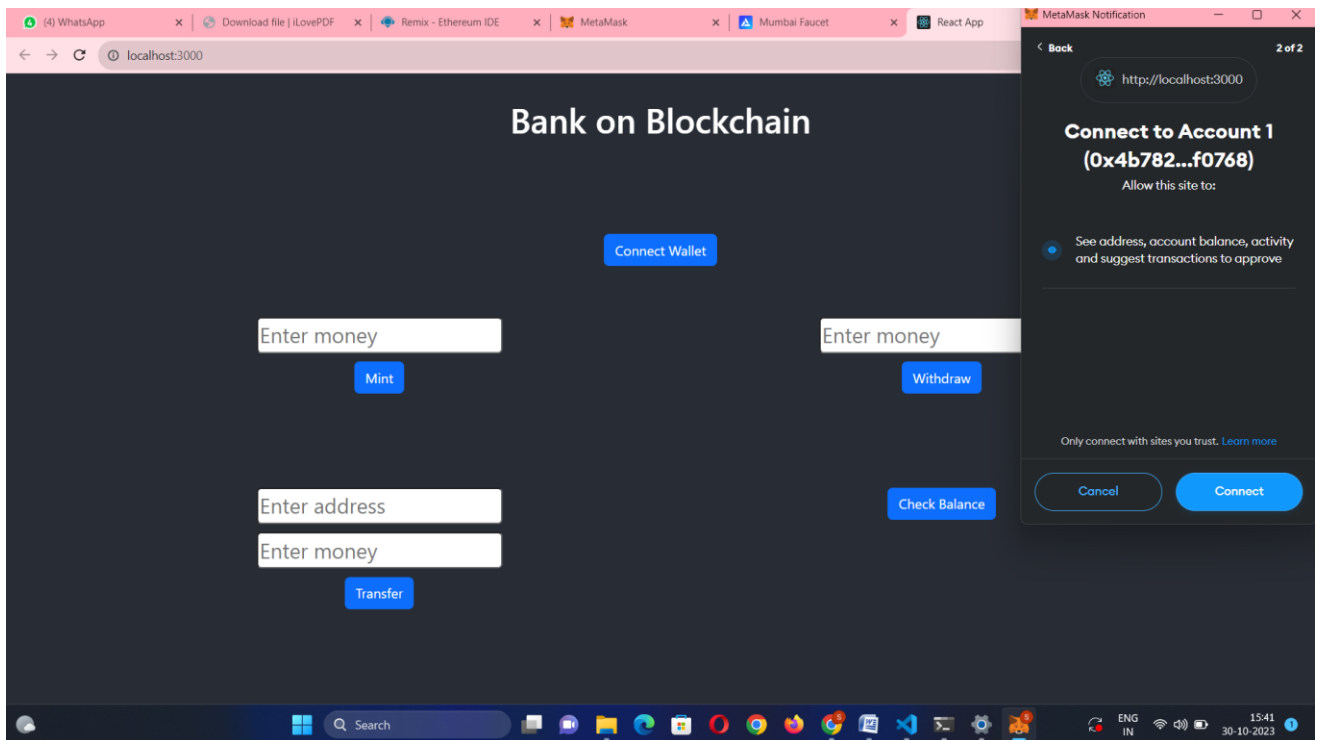
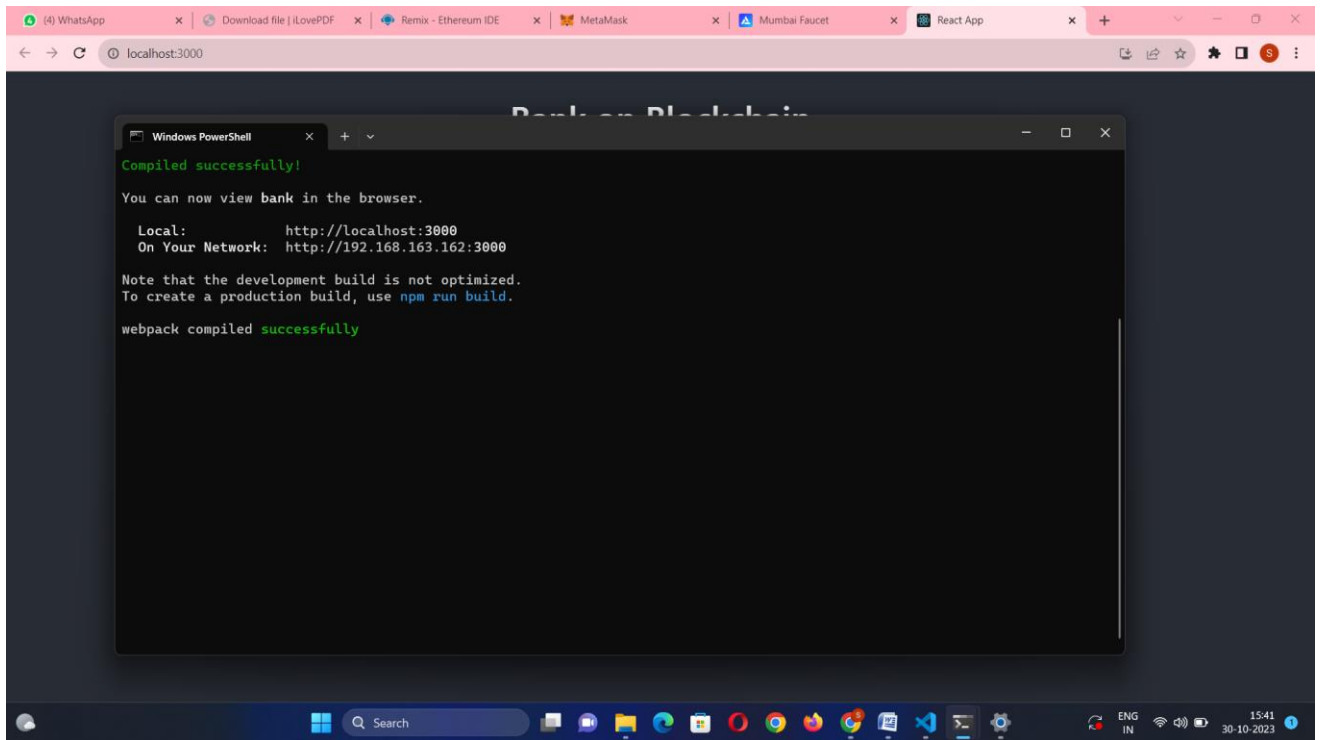
Activity

\$0.00 USD

34°C Rain showers

Search

ENG IN 15:34 29-10-2023



10.ADVANTAGES & DISADVANTAGES:

Advantages:

- **Transparency:** Transactions and data are recorded on an immutable, transparent ledger, allowing for real-time auditing and verification of financial activities.
 - **Security:** Blockchain's cryptographic features and decentralized architecture enhance the security and privacy of user data and assets, reducing the risk of fraud and cyberattacks.
 - **Reduced Costs:** Automation and elimination of intermediaries lead to cost savings in transaction processing and settlement.
 - **Efficiency:** Transactions are processed quickly and in a trustless manner, reducing settlement times and improving the overall efficiency of the financial system
- Smart Contracts:** Automation of financial processes through smart contracts enables self-executing agreements, reducing the need for manual intervention and minimizing errors.

Disadvantages:

- **Stability Issues:** Blockchains can face scalability challenges, leading to slower transaction processing times and higher fees during peak usage.
- **Complexity:** Developing and maintaining smart contracts can be complex and require expertise in blockchain technology, potentially increasing development costs.
- **Irreversible transactions:** Once transactions are confirmed on the blockchain, they are typically irreversible, which can be problematic in cases of errors or disputes.

11. CONCLUSION:

In conclusion, central bank smart contracts using blockchain technology hold the potential to revolutionize the traditional banking system by providing enhanced security, transparency, and efficiency. These innovative solutions offer a promising path towards a more inclusive and accessible financial ecosystem, reducing costs, streamlining processes, and promoting financial empowerment. However, they also come with their set of challenges, including scalability issues, regulatory complexities, and security concerns that must be carefully addressed. The successful implementation of central bank smart contracts in a real-world financial context will depend on continued advancements in technology, regulatory adaptation, and user education. While the road ahead may be complex, the transformative possibilities of blockchain in central banking make it a promising area of exploration for the financial industry and regulators.

12. FUTURE SCOPE:

The future scope of central bank smart contracts using blockchain technology is promising and holds the potential for transformative changes in the financial industry. Here are some key areas of future development. Central bank smart contracts can facilitate greater financial inclusion by offering banking services to underserved populations, particularly in regions with limited access to traditional banking. Efforts are underway to improve interoperability between different blockchain networks and legacy financial systems, enabling seamless cross-border transactions and greater integration with existing infrastructure. The use of blockchain technology for asset tokenization, including real estate, stocks, and commodities, will likely become more prevalent, enhancing liquidity and accessibility to a wider range of assets.

Blockchain-based systems are expected to streamline and reduce the cost of cross-border payments, benefitting international trade and remittances. As the technology matures, regulators are likely to develop more comprehensive and tailored regulatory frameworks for central bank smart contracts, ensuring legal compliance while promoting innovation. Central banks worldwide are exploring the development of central bank digital currencies (CBDCs) to complement existing fiat currencies, offering a secure and efficient digital means of exchange. Blockchain's flexibility and programmable nature enable the creation of innovative financial products, such as decentralized finance (DeFi) applications, which are expected to evolve and diversify. Privacy-enhancing technologies will likely be integrated into blockchain systems to address concerns about data privacy while maintaining transparency. The future of central bank smart contracts using blockchain is marked by ongoing innovation, regulatory evolution, and a shift towards a more inclusive and efficient financial ecosystem. While challenges exist, the opportunities for enhancing financial services, reducing costs, and increasing access to banking are substantial, making this a dynamic and evolving field in the financial industry.

13. APPENDIX

Source Code:

SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Bank {

address public owner;

mapping(address => uint256) public balances;

constructor() {

owner = msg.sender;

}

modifier onlyOwner() {

require(msg.sender == owner, "Only contract owner can call this");

_;

}

function mintMoney(uint256 amount) external onlyOwner {

require(amount > 0, "Amount must be greater than 0");

balances[msg.sender] += amount;

}

function withdrawMoney(uint256 amount) external {

require(balances[msg.sender] >= amount, "Insufficient balance");

balances[msg.sender] -= amount;

}

function transferFunds(address payable receiptAddress,uint _amount)

public onlyOwner{

require(balances[msg.sender] >= _amount, "Insufficient balance");

balances[msg.sender] -= _amount;

balances[receiptAddress] += _amount;

```
}  
  
function checkBalance() external view returns (uint256) {  
    return balances[msg.sender];  
}  
}
```

13.1 GitHub & Project DemoLink:

<https://github.com/Sangeethakumarselvi>